МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ПЕДАГОГІЧНИЙ УІВЕРСИТЕТ імені М. П. ДРАГОМАНОВА

На правах рукопису

Костюченко Андрій Олександрович

УДК 378: 372.8: 004.415.25

Комп'ютерно орієнтована методична система підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів

13.00.02 – теорія та методика навчання (інформатика)

Дисертація
на здобуття наукового ступеня
кандидата педагогічних наук

Науковий керівник:
Горошко Юрій Васильович,
доктор педагогічних наук, доцент

Київ – 2014

ЗМІСТ

Перелік умовних позначень..... 4

Вступ 5

Розділ 1. Науково-методичні засади навчання програмування в педагогічних ВНЗ 14

1.1. Аналіз педагогічних досліджень щодо навчання програмування 14

1.2. Психолого-педагогічні складові готовності майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів..... 29

1.3. Концептуальні засади розроблення програмних засобів 36

1.3.1. Парадигма програмування..... 36

1.3.2. Підходи до проектування ПЗ..... 46

1.3.3. Технології програмування 58

1.4. Особливості проектування педагогічних програмних засобів 68

1.4.1. Вимоги до ППЗ..... 68

1.4.2. Показники якості ППЗ 73

1.4.3. Етапи проектування та створення педагогічного програмного комплексу 79

1.5. Науково-методичні підходи до навчання програмування 81

1.5.1. Задачний підхід 83

1.5.2. Ітераційно-поступальний підхід..... 85

1.5.3. Підхід парного програмування..... 86

1.5.4. Підхід колективного програмування 88

Висновки до розділу 1..... 89

Розділ 2. Компоненти методичної системи підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів..... 91

2.1. Загальна методика дослідження проблеми 91

2.2. Методи навчання створення педагогічних програмних засобів 93

2.2.1. Рівнева диференціація навчання 98

2.3. Пропедевтика навчання студентів створення ППЗ 102
2.4. Елементи навчально-методичного комплексу з дисципліни “Технології програмування та створення педагогічних програмних засобів” 104
2.4.1. Особливості математичних ППЗ 109
2.4.2. Призначення та особливості інтерфейсу ППЗ Gran2d 118
2.4.3. Методика навчання розробки ППЗ з геометрії на прикладі ППЗ Gran2d 133
2.4.4. Призначення та особливості інтерфейсу ППЗ Numet 158
2.4.5. Методика навчання розробки ППЗ з чисельних методів математики на прикладі ППЗ Numet 166
Висновки до розділу 2 188
Розділ 3. Аналіз ефективності розроблених компонентів методичної системи 191
3.1. Підхід експертного оцінювання для визначення якості підготовки студентів до розроблення ППЗ 192
3.2. Практична реалізація та аналіз результатів експериментального дослідження 196
Висновки до розділу 3 203
Висновки 205
Список використаних джерел 209
Додатки 239

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ВНЗ	вищий навчальний заклад
ЕОМ	електронна обчислювальна машина
ІКТ	інформаційно-комунікаційні технології
ООП	об’єктно-орієнтоване програмування
ОС	операційна система
ПДСК	прямокутна Декартова система координат
ПЗ	програмний засіб
ППЗ	педагогічний програмний засіб
ППК	педагогічний програмний комплекс
СЛАР	система лінійних алгебраїчних рівнянь

ВСТУП

Актуальність теми. З появою в школі комп'ютерів почали створюватися комп'ютерні програми, призначені для підтримки навчання шкільних предметів. За такими програмами закріпився термін „педагогічні програмні засоби".

Педагогічні програмні засоби (ППЗ) – це комп'ютерна програма чи сукупність комп'ютерних програм, призначених для комп'ютерної підтримки навчання та досягнення конкретних навчальних цілей [34, с. 5]. В науковій літературі поняття педагогічний програмний засіб інколи замінюється поняттям програмний засіб навчального призначення. ППЗ є головною частиною педагогічного програмного комплексу (ППК) навчання. ППК включає в себе, крім ППЗ, методичні та дидактичні матеріали і настанови, розраховані на використання вказаного ППЗ для комп'ютерної підтримки навчально-пізнавальної діяльності.

Однією з проблем впровадження ППЗ в навчальний процес є те, що досить часто ППЗ створюються програмістами, мало зв'язаними зі школою або вчителями-предметниками з недостатнім досвідом створення ПЗ.

В одному випадку ППЗ створюється досвідченими програмістами, досить часто далекими від галузі освіти. Найчастіше такий ПЗ спочатку створюється, не зважаючи на вимоги, що висуваються до ППЗ, після чого розпочинаються способи знайти йому місце в навчальному процесі. Таким чином такі програми створюються частіше за все не так, як це потрібно учню та вчителю, а так, як це зручно і зрозуміло розробнику. При цьому, з одного боку в результаті використання надмірно складних програм певною мірою зменшується ефективність навчання, а з іншого – надмірна кількість інструментарію, що пропонуються користувачеві в цих програмах, занадто перевищує потреби учнів для розв'язування поставлених перед ними завдань.

В іншому випадку, коли ППЗ створюється вчителем-предметником, мало обізнаним з особливостями використання та застосування мов програмування, програми виявляються "безпорадними" з точки зору характеристик вибраного

середовища програмування. В них використовується комп'ютер не завжди ефективно, проте найчастіше в їх основу покладена адекватна методика навчання предмета і програми часто дидактично досить коректно обґрунтовані.

Як показав аналіз існуючих ППЗ, можна з впевненістю сказати, що лише невелика їх частина може ефективно використовуватися при навчанні різних предметів у школі.

Зважаючи на сказане, вбачається два шляхи створення сучасних ППЗ. У першому випадку програмуванням займається професійний програміст при активній участі педагогів з тієї предметної галузі, до якої відноситься ППЗ. В цій ситуації володіння педагогом основами принципами створення ППЗ дозволить підвищити ефективність його взаємодії з програмістами, прискорити процес розробки. У другому випадку сам педагог, який оволодів компетентностями в галузі створення ППЗ, зможе його розробляти. Проте варто зауважити, що створення ППЗ в загальному випадку потребує колективної праці не тільки вчителів-предметників та програмістів, але і психологів, гігієністів, дизайнерів.

Зрозуміло, що на даному етапі досить серйозне навчання програмування можливе тільки на тих спеціальностях у педагогічних ВНЗ, де інформатика є основною спеціальністю, або спеціалізацією.

Окремі аспекти навчання програмування розглядають у своїх роботах: Анохін В. Є. [43], Бакуменко К. В. [11], Вернигоренко С. А. [31], Глинський Я. М. [42; 43], Гришко Л. В. [58; 59; 61; 229], Гуржій А. М. [214], Дорошенко Ю. О. [93-95], Жалдак М. І. [82; 84], Жужжалов В. Е. [90; 91], Завадський І. О. [93-95], Зарецька І. Т. [97], Зеленяк О. П. [99], Ільченко А. А. [103], Караванова Т. П. [107], Колодяжний Б. Г. [97], Лукаш І. М. [194; 195], Львов М. С. [214], Меджитова Л. М. [157], Морзе Н. В. [82; 160], Осипова Н. В. [11], Потапова Ж. В. [93-95], Рамський Ю. С. [82; 194; 195; 196], Ряжська В. А. [43], Сейдаметова З. С. [201], Семеріков С. О. [202], Співаковський О. В. [11; 211; 212; 214], Спірін О. М. [216], Триус Ю. В. [229], Цибко Г. Ю. [196], Шевчук П. Г. [241; 242] та інші.

Різноманітні аспекти вибору та використання середовища програмування досліджують вітчизняні та зарубіжні науковці: Волошинов С. А. [35],

Габрусєв В. Ю. [49], Глинський Я. М. [43], Горошко Ю. В. [54; 56], Гришко Л. В. [59], Жалдак М. І. [86], Завадський І.О. [95], Морзе Н. В. [160; 162], Онищенко С. М. [169; 170], Рамський Ю. С. [193; 196], Сейдаметова З. С. [201], Семеріков С. О. [203], Співаковський О. В. [214], Теплицький І. О. [203; 224], Триус Ю. В. [229], Франчук В. М. [170], Цибко Г. Ю. [196] та інші.

Проблеми впровадження об'єктно-орієнтованої парадигми навчання програмування розглядаються у роботах Вернигоренка С. А. [31], Горошка Ю. В. [53], Лесневского А. С. [145], Лукаш І. М. [194; 195], Петрова А. Н. [176], Рамського Ю. С. [194; 195], Семерікова С. О. [202], Шевчука П. Г. [242], та інших.

Питаннями присвяченими розробці ППЗ, займаються Волошинов С. А. [36], Горошко Ю. В. [53], Жалдак М. І. [87], Зубкова Т. М. [100], Пеньков А. В. [175], Раков С.А. [190] та інші.

Методичні та дидактичні проблеми застосування комп'ютера як засобу навчально-пізнавальної діяльності, психолого-педагогічні аспекти використання інформаційних технологій навчання, в тому числі і ППЗ, в навчальному процесі розглядаються в роботах Ю. В. Горошка [53], М. І. Жалдака [80; 81; 84; 85; 87], В. І. Клочка [111], Т. П. Кобильника [115; 116; 117], А. В. Пенькова [175], Ю. С. Рамського [192], С. А. Ракова [188; 189; 191], О. І. Скафа [33; 207], Є. М. Смирнової [210], Ю. В. Триуса [228] та ін.

З огляду на сказане постає задача навчити майбутніх учителів відповідних спеціальностей у педагогічних ВНЗ не просто програмувати, а і створювати ППЗ для підтримки навчання свого, а можливо і інших предметів. Оскільки при створенні такого ПЗ розробник має не лише володіти предметом, в галузі якого відбувається розробка, але і вміти програмувати, йому необхідно розумітися на дидактичних та методичних засадах використання ППЗ.

Разом із тим поза увагою дослідників залишаються питання підготовки майбутніх учителів математики та інформатики до розроблення ППЗ. Актуальність та необхідність теоретичного та практичного розв'язання даної проблеми дослідження зумовило вибір теми дисертації **«Комп'ютерно**

орієнтована методична системи підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів».

Зв'язок роботи з науковими програмами, планами, темами. Тема дослідження входить до плану науково-дослідної роботи Інституту інформатики НПУ імені М.П. Драгоманова та входить в рамки науково-дослідної теми Чернігівського національного педагогічного університету імені Т.Г.Шевченка «Розробка компонентів методичної системи навчання інформатики студентів педагогічних ВНЗ в умовах використання вільного програмного забезпечення» (реєстраційний номер 00112U001073).

Тема дослідження затверджена на засіданні Вченої ради Національного педагогічного університету імені М.П. Драгоманова (протокол № 12 від 26 квітня 2013 р.) та узгоджена в Міжвідомчій раді з координації наукових досліджень з педагогіки та психології в Україні при АПН України (протокол № 5 від 28 травня 2013 р.).

Метою дослідження є розробка науково обґрунтованих компонентів комп'ютерно орієнтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів, яка сприяє активізації навчально-пізнавальної, дослідницької діяльності студентів, розкриттю їх творчого потенціалу, розвитку їхньої самостійності та індивідуальних здібностей й ґрунтується на широкому впровадженні в навчальний процес новітніх педагогічних та інформаційно-комунікаційних технологій.

Об'єкт дослідження – процес навчання інформатичних дисциплін у вищих педагогічних навчальних закладах III-IV рівнів акредитації.

Предмет дослідження – компоненти комп'ютерно орієнтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів.

Гіпотеза дослідження ґрунтується на припущенні, що цілеспрямоване використання науково обґрунтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних

засобів, що базується на використанні вихідних кодів реально використовуваних ППЗ, буде сприяти: активізації пізнавальної діяльності студентів; набуттю практичних навичок створення реальних ППЗ; опануванню особливостей аналізу і опрацювання математичних об’єктів; покращенню міжпредметних зв’язків, змістовної єдність програмування з іншими навчальними предметами, для яких будуть створюватися ППЗ.

Відповідно до мети дослідження необхідно було вирішити такі **завдання**:

1. Здійснити аналіз проблеми навчання програмування у педагогічних ВНЗ.

2. Проаналізувати вітчизняну та зарубіжну психолого-педагогічну, науково-методичну літературу з метою вивчення стану дослідженості проблеми навчання майбутніх учителів математики та інформатики проектування та створення ППЗ. На основі отриманих результатів визначити концептуальні засади, виокремити особливості та етапи проектування та створення ППЗ, а також визначити умови та можливі шляхи реалізації процесу підготовки майбутніх учителів математики та інформатики до розроблення ППЗ.

3. Визначити і обґрунтувати вибір мови та середовища програмування, що використовуватиметься для підготовки майбутніх учителів математики та інформатики до розроблення ППЗ.

4. Розробити компоненти комп’ютерно орієнтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення ППЗ, що базуватиметься на використанні вихідних кодів реально використовуваних ППЗ Numet та Gran2d.

5. Обрати підхід перевірки результатів навчальної діяльності майбутніх учителів математики та інформатики, на основі якого можна було б перевірити ефективність розроблених компонентів методичної системи навчання.

6. У ході педагогічного експерименту перевірити ефективність розроблених компонентів комп’ютерно орієнтованої методичної системи підготовки підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів.

Для розв’язання поставлених завдань застосовувались такі **методи досліджень**:

– *теоретичні*: системний науково-методологічний аналіз чинних стандартів освіти (1.1, 2.3, 2.4, 3.1 (тут і далі підрозділи дисертації)), навчальних програм, підручників і навчальних посібників (1.1, 1.5, 2.2, 2.4), монографій, дисертаційних досліджень, статей і матеріалів науково-методичних конференцій з проблеми дослідження та застосування сучасних інформаційно-комунікаційних технологій в навчальному процесі (1.1-1.5, 2.2, 2.4);

– *емпіричні*: спостереження навчального процесу; аналіз результатів навчання студентів у відповідності до проблеми дослідження; анкетування, тестування, бесіди зі студентами та вчителями; аналіз роботи учителів за основними положеннями дослідження (1.5, 2.1, 2.3, 2.4, 3.1, 3.2);

– *педагогічний експеримент* для з’ясування педагогічної ефективності окремих компонентів розроблюваної комп’ютерно орієнтованої методичної системи навчання (3.1, 3.2);

– *методи математичної статистики для опрацювання даних експерименту:* статистичне опрацювання результатів дослідження; визначення кількісних і якісних показників (3.2).

Наукова новизна одержаних результатів дисертаційного дослідження полягає в тому, що:

– розроблені, теоретично обґрунтовані та експериментально перевірені компоненти комп’ютерно орієнтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення ППЗ;

– визначено психолого-педагогічні основи методичної системи підготовки майбутніх учителів математики та інформатики до розроблення ППЗ та методи, що можуть бути використані в процесі їх навчання створювати ППЗ;

– визначено перелік критеріїв, за якими має проводитися експертне оцінювання якості створюваних ППЗ;

– вдосконалено підходи до навчання об’єктно-орієнтованого програмування в педагогічних ВНЗ;

– дістали подальшого розвитку підходи до використання вільно поширюваного програмного забезпечення при навчанні програмування.

Практичне значення результатів дисертаційного дослідження полягає в тому, що:

1) *обґрунтовано*:

– цілі навчання і зміст дисципліни “Технології програмування та створення ППЗ” на основі використання вихідного коду вже розроблених ППЗ;

– важливість підготовки майбутніх учителів математики та інформатики до створення ППЗ і подальшого їх використання в навчальному процесі;

2) *вдосконалено* ППЗ Gran2d для автоматизації досліджень планіметричних об’єктів для ОС Windows, який за допомогою технологій віртуалізації або емуляції можна використовувати при роботі з усіма сучасними ОС;

3) *створено*:

– ППЗ Numet (для ОС Windows) для підтримки навчання чисельних методів в педагогічних ВНЗ;

– компоненти комп’ютерно орієнтованої методичної системи навчання дисципліни “Технології програмування та створення ППЗ”, які подані в навчальному посібнику “Теорія і методика розробки педагогічних програмних засобів”, виданому у співавторстві з Горошком Ю.В.

– систему лабораторних завдань з дисципліни “Технології програмування та створення ППЗ” для студентів математичних спеціальностей педагогічних університетів;

4) *запропоновано* внести зміни до змісту курсів навчальних дисциплін, в межах яких відбувається вивчення об’єктно-орієнованого програмування, для пропедевтики підготовки до створення ППЗ.

Впровадження результатів дослідження в педагогічну практику підтверджується довідками: Чернігівського національного педагогічного університету імені Т. Г. Шевченка (№ 36 від 08.11.2013 р.), Національного педагогічному університеті імені М. П. Драгоманова (№07-10/2683 від 18.11.2013 р.), Уманського державного педагогічного університету

імені Павла Тичини (№ 1910/01 від 30.10.2013 р.), Криворізького педагогічного інституту при Криворізькому національному університеті (№ 02/19/02-356/03 від 22.10.2013 р.), Чернігівського обласного інституту післядипломної педагогічної освіти імені К.Д. Ушинського (№ 01-12/1128 від 18.10.2013 р.), Ніжинського державного університету імені Миколи Гоголя (№ 04 від 01.11.2013 р.)

Особистий внесок здобувача. У працях, опублікованих у співавторстві, автору належать:

– визначення переваг та підходів до використання ППЗ Gran2d при виконанні геометричних перетворень в навчальному процесі навчання планіметрії в публікації [32] у співавторстві з Є.Ф. Вінниченком;

– розгляд ідеї та алгоритму розв'язування задачі обласної учнівської олімпіади з програмування 2007 року в публікації [51] у співавторстві з Ю.В. Горошком;

– огляд парадигм та технологій програмування, визначення окремих вимог та критеріїв якості ППЗ, розділ про структуру ППЗ Gran2d в роботі [53] у співавторстві з Ю.В. Горошком;

– добір та опис вправ стосовно застосування розробленого ППЗ Numet подано в публікації [55] у співавторстві з Ю.В. Горошком;

– визначення ролі та місця вільно поширюваних офісних пакетів та середовищ програмування в навчальному процесі в публікаціях [54; 56] у співавторстві з Ю.В. Горошком, М.І. Шкардибардою;

– аналіз окремих комп'ютерних програм для тестування, огляд переваг та недоліків їх використання в навчальному процесі в публікації [74] у співавторстві з К.М. Дубиною;

– розробка програмного засобу з урахуванням дидактичних цілей та вимог щодо його використання подано в публікації [128] у співавторстві з Г.О. Михаліним, С.Л. Надточій;

– робота над інтерфейсом, програмна реалізація ППЗ Numet [181].

– робота над інтерфейсом, програмна реалізація нових та вдосконалення вже існуючих опцій в розробці ППЗ Gran2d [180];

Апробація результатів дисертації. Основні положення і результати дослідження доповідались та обговорювались на наукових конференціях різного рівня: міжнародній конференції пам'яті проф. І.І.Мархеля «Нові інформаційні технології в навчальних закладах України» (Одеса, Одеський національний морський університет, 21-26 червня 2005 р.), науково-практичній конференції «Вільне програмне забезпечення в освіті, науці, бізнесі» (Чернігів, ЧДТУ, 13-15 травня 2010 р.), науково-практичній конференції «Вільне програмне забезпечення в освіті, науці, бізнесі» (Чернігів, ЧДТУ, 19-20 травня 2011 р.), міжнародній науково-практичній конференції молодих науковців «Інформаційні технології як інноваційний шлях розвитку України у XXI столітті» (Ужгород, Закарпатський державний університет, 06-08 грудня 2012 р.), всеукраїнській науково-практичній конференції "Інформаційні та моделюючі технології" (ІМТ-2013) (Черкаси, 17–19 травня 2013 р.), IX Международная конференция "Стратегия качества в промышленности и образовании " (г. Варна, Болгария, 31 мая – 7 июня 2013 г.).

Матеріали і результати дослідження обговорювалися на засіданнях і семінарах кафедри теоретичних основ інформатики Національного педагогічного університету ім. М.П. Драгоманова, кафедри інформатики і обчислювальної техніки Чернігівського національного педагогічного університету імені Т.Г.Шевченка, а також апробовані шляхом публікацій.

Публікації. З досліджуваної проблеми дисертаційного дослідження опубліковано 20 наукових праць, серед них: 1 навчально-методичний посібник для вчителів та студентів, 12 статей у наукових фахових виданнях України (6 одноосібних), 2 одноосібні статті в наукових виданнях зарубіжних країн, 1 стаття у науково-методичному журналі в співавторстві, 4 тези доповідей – у матеріалах конференцій (3 одноосібних), 2 програмні засоби (1 одноосібний).

Структура та обсяг дисертації. Дисертація складається з переліку умовних позначень, вступу, трьох розділів, висновків, списку використаних джерел (264 найменування, з них 18 іноземними мовами), 8 додатків (обсягом 44 сторінок). Загальний обсяг дисертації 283 сторінок, з них 205 сторінки основного тексту. Робота містить 23 рисунків і 8 таблиць.

РОЗДІЛ 1
НАУКОВО-МЕТОДИЧНІ ЗАСАДИ НАВЧАННЯ ПРОГРАМУВАННЯ В ПЕДАГОГІЧНИХ ВНЗ

1.1. Аналіз педагогічних досліджень щодо навчання програмування

Одним із перших, хто в 70-80 роках минулого століття почав працювати над питанням методики навчання інформатики, був академік А. П. Єршов. Його відомі праці “Програмування – друга освіта” та “Звідки беруться люди, що можуть створювати надійне програмне забезпечення” створили фундамент для методики навчання інформатики як науки. Подальше активне обговорення питань навчання інформатики призвело до створення та впровадження державних освітніх стандартів, що відобразили існуюче на той час бачення предмета та відповідних йому знань.

Одним із найбільш актуальних питань у навчанні інформатики є роль та місце програмування в навчальному плані. У своїй роботі Кривонос О. М. [136, с. 135] зазначає «Виникла необхідність під час навчання програмуванню використовувати методи підвищеної мотивації навчання студентів комп’ютерних спеціальностей та майбутніх викладачів інформатики для школи». Поняття програмування [229] визначається, як один із видів людської діяльності, що потребує надзвичайної точності, педантичності, алгоритмічного стилю мислення. “Прищеплення” студентам цих якостей є найбільш складним у процесі навчання програмування.

За роки свого існування більшість курсів з алгоритмізації і програмування фокусувалися переважно на формуванні навичок програмування. У результаті аналізу державних стандартів вищої освіти, навчальних планів і робочих навчальних програм зі спеціальностей, де здійснюється підготовка фахівців у галузі інформатики, комп’ютерної та програмної інженерії, інформаційно-телекомунікаційних технологій визначено, що [59; 136]: курс з алгоритмізації і програмування базується на концепціях, які важливі для практики програмування незалежно від парадигми програмування, що використовується у навчальному

процесі; вміння програмувати є обов’язковим для всіх студентів, що навчаються за напрямком "Інформатика". курс з алгоритмізації і програмування є вступним курсом і повинен вивчатися на молодших курсах.

Як вказано в законі України “Про основні засади розвитку інформаційного суспільства в Україні на 2007–2015 роки” [186] «Навчання програмування – окремий вектор сучасної освіти. Загальні знання з програмування обов’язковий компонент інформаційно-комунікаційної компетенції сучасного фахівця високого рівня. Необхідність навчання програмування продиктована завданнями сучасної освіти стосовно підготовки молоді до життя та діяльності в умовах інформаційного суспільства». У своїй роботі [59, с. 136] Гришко Л.В. зазначає: «Головною метою навчання основ програмування є формування у майбутнього фахівця з комп’ютерних наук компетентностей з алгоритмізації і програмування, розкриття його творчого потенціалу, розвиток самостійності та індивідуальних здібностей особистості студента». В свою чергу Крамар Ю.М. [133, с. 54] метою програмування вважає «… створення програми, що відповідає певним вимогам. Успішне досягнення цієї мети залежить від того, наскільки повно виконано вимоги. Серед цих вимог є і вимоги до тексту програми, які відображують стиль. Склалося так, що вимоги виражаються в правилах, що використовує програміст. Отже, стиль програмування можна описати шляхом формулювання набору правил». В роботі [157, с. 52] Меджитова Л.М. визначає, що мета навчання програмування «… полягає у підготовці базису для оволодіння і розвитку умінь пошуку ефективних способів розв’язування задач з наступною їх програмною реалізацією» та зазначає, що «Для досягнення поставленої мети необхідно сформувати у студентів вміння розробляти алгоритми і записувати їх в різних формах, виділяти і описувати об’єкти та їх взаємозв’язки, читати і розуміти готові алгоритми, записувати програмний код, розуміти семантику основних структур управління в програмному коді, ефективно використовувати інтегроване програмне середовище в ході розв’язування задач, володіти термінологією». Співаковський О.В. [214, с. 50] вказує, що «Розробка і наукове обґрунтування різних стилів програмування – одне з основних завдань теорії програмування».

Варто вказати, що на значення навчання алгоритмізації і програмування для розвитку деяких додаткових навиків вказували дослідники, які вивчали систему підготовки професійних програмістів. Так, З. С. Сейдаметова відзначає, що для випускників-бакалаврів комп'ютерних наук важливі і деякі додаткові навички, які можуть бути сформовані в контексті комп'ютерних наук, проте мають загальний характер і корисні також і в багатьох інших контекстах [201, с. 81]. С. А. Волошин, розглядаючи значення навчання програмування під час підготовки майбутніх судноводіїв зазначає: «алгоритмічна підготовка сучасного фахівця за спеціальністю «Судноводіння» сприяє розвитку динамічності мислення, його гнучкості, формуванню уміння розділяти складний об'єкт на прості складові, визначати взаємозв'язки між ними. Все це необхідно для вивчення і побудови формальних моделей в будь-якій наочній області і надає можливість навчитися такому підходу до будь-якого завдання, при якому його рішення виступає як об'єкт конструювання і винаходу» [35, с. 104].

Окрім того вагоме місце в навчанні програмування займає використання міжпредметних зв'язків. Так у своїй роботі [116, с. 1] Т. П. Кобильник вказує «Використання міжпредметних зв'язків в навчальному процесі стимулює студентів до професійного вдосконалення. За допомогою багатосторонніх міжпредметних зв'язків не тільки на якісно новому рівні вирішуються завдання навчання, розвитку і виховання студентів, але й закладається фундамент для комплексного бачення і вирішення складних проблем реальної дійсності».

Традиційно вважається, що в основі теорії програмування лежить поняття алгоритму. Зазвичай знайомство з програмуванням розпочинають саме з вивчення цього поняття. Історію виникнення терміну «Алгоритм» детально описав Кнут [113, с. 19-20]. Він же дав визначення властивостей алгоритму [113, с. 23-25]. Нині існує багато підходів до визначення поняття алгоритму та його використання у навчанні [137; 145; 155]. Зокрема, досліджуючи це поняття у різних його трактуваннях, Копаєв О. В. визначає алгоритм, як модель алгоритмічного процесу. «Алгоритм і алгоритмічний процес – це різні об'єкти, причому вони мають різну природу. Разом з тим, вивчаючи алгоритм, можна

зробити й певні висновки про алгоритмічний процес, який він задає. Адже алгоритм повністю детермінує алгоритмічний процес» [119, с. 190].

Деякі науковці та педагоги на перше місце навчання алгоритмізації і програмування ставлять навчання власне алгоритмізації. Так у своїй роботі [160, с. 3] Морзе Н.В. визначає: «Мета навчання алгоритмізації – сформувати знання та вміння щодо основних способів організації операцій і даних, а також застосування базових алгоритмічних конструкцій при складанні описів алгоритмів розв'язування різноманітних задач» та вказує, що під час вивчення основ алгоритмізації увага повинна приділятися насамперед «виявленню загальних закономірностей і принципів алгоритмізації; основним етапам розв'язування задач за допомогою сучасних інформаційних технологій; аналізові поставленої задачі, методам формалізації та моделювання реальних процесів та явищ; добору виконавця поставленої задачі, виходячи з того, що він є також певним об'єктом із притаманними йому властивостями й набором допустимих операцій, які слід аналізувати з метою правильного та ефективного їх використання; методам та засобам формалізованих описів дій виконавця, сучасним засобам їх конструювання та реалізації за допомогою комп'ютера». На думку А.П.Єршова, навчальна алгоритмічна мова, яку він запропонував, виконує дві основні функції у навчанні алгоритмізації. По-перше, за її допомогою можна стандартизувати, надати єдиної форми описам всіх алгоритмів, які розглядаються у шкільному курсі інформатики, що важливо для розуміння суті алгоритмізації формування уявлень про властивості алгоритмів. По-друге, вивчення навчальної алгоритмічної мови є пропедевтикою вивчення мов програмування. Простота конструкцій навчальної алгоритмічної мови і правил їх використання надає можливість успішно застосовувати цю мову на початковому етапі навчання програмування. Базові конструкції і правила лежать в основі багатьох мов програмування. Тому опанування навчальною алгоритмічною мовою дозволить надалі легко перейти до використання реальних мов програмування. С.О.Семеріков вважає [179, с. 6]: "Процес алгоритмізації розглядається через побудову алгоритмічних моделей, природним продовженням яких є програмні моделі".

У ході досліджень [59; 229; 215; 86; 61] визначено основні функції навчання алгоритмізації і програмування: загальноосвітня, розвивальна і виховна функції.

Загальноосвітня функція навчання алгоритмізації і програмування полягає в тому, щоб забезпечити: формування у студентів знань про фундаментальні концепції і парадигми програмування; методологію побудови інформаційних моделей; модульний принцип розробки програм; типові алгоритмічні конструкції; методи структурного низхідного програмування; методи побудови алгоритмів; критерії ефективності алгоритмів і програм; засоби для розробки, супроводу та організації багатофайлових прикладних програм; формування у студентів компетентностей практичного програмування мовами програмування.

Розвивальна функція навчання алгоритмізації і програмування полягає в тому, щоб сприяти розвитку у студентів алгоритмічного стилю мислення, інтелектуальних якостей і творчих здібностей, які необхідні для розв’язування задач програмування; формуванню у студентів здатності бачити задачу одночасно на різних рівнях деталізації, узагальнювати типові ситуації при вивченні нових мов програмування, навчатися протягом усього життя; формуванню у студентів розуміння зв’язку теорії програмування з практикою програмування; розуміння галузей застосування здобутих компетентностей з програмування в інших професійно-орієнтованих і спеціальних дисциплінах та в сфері майбутньої професійної діяльності.

Виховна функція навчання алгоритмізації і програмування полягає у формуванні у студентів таких якостей як педантичність, дисциплінованість, акуратність, внутрішня керованість, наполегливість, усвідомлення особистої відповідальності за результати своєї праці, прагнення до самоутвердження через творчу діяльність, уміння працювати як індивідуально, так і в команді, системність і цілеспрямованість у навчальній та професійній діяльності.

У роботах Гришко Л.В. [59; 61] визначено принципи навчання програмування, а саме «спрямованості процесу навчання на всебічний, гармонійний розвиток особистості; науковості і доступності навчання; створення відповідних умов для функціонування процесу навчання; систематичності і

послідовності навчання; свідомості, активності і самостійності студентів у навчанні при керівній ролі викладача; індивідуалізації і колективності навчання; ефективного поєднання методів навчання; міцності і дієвості знань, умінь і навичок; забезпечення оперативного контролю і самоконтролю в навчанні».

Що стосується підходів і методів навчання програмування, то у статті «A New Approach to Teaching Programming» [250] описується підхід, заснований на трьох припущеннях, які відображають бачення авторів процесу навчання програмування. Перше припущення полягає в необхідності інтеграції теорії та практики. Друге припущення авторів полягає в необхідності зміщення уваги від написання коду до проектування розв'язку задачі. Третє припущення – багатогранність процесу розробки програмного забезпечення. Для розв'язування різних задач можуть знадобитися зовсім різні підходи. У зв'язку з цим автори переконані в тому, що програміст повинен володіти різними парадигмами.

В свою чергу пропозиція авторів статті «Teaching programming using visualization» [252] полягає у використанні прикладів з реального світу для пояснення понять і концепцій. Автори роботи підкреслюють, що програмування по суті своїй ґрунтується на абстракціях і математиці, а це викликає труднощі у студентів молодших курсів. І дійсно, при вивченні програмування перед студентом постає завдання розглянути реальний об'єкт або перебіг процесу (наприклад, перехід людини через дорогу) і описати алгоритм. У зв'язку з цим одним з основних припущень авторів роботи [252] є використання мультимедіа в навчальному процесі, включаючи анімацію, звук і відео, за допомогою яких вивчаються, наприклад, структури управління ходом розв'язування.

У своїх роботах Триус Ю.В. [229] також звертає увагу на підходи та методи навчання програмування та визначає такі з них: задачний підхід, диференційований підхід, паралельне навчання, парне програмування, проблемний метод навчання, методи інтерактивного та активного навчання.

Корисний ефект для навчання програмування дає процес візуалізації виконання алгоритмів. Так у статті "Проведення обчислювального експерименту засобами дистанційного вивчення курсу «Основи алгоритмізації та

програмування»” автори Співаковський О.В., Осипова Н.В., Львов М.С. [11, с. 15] зазначають, що «використання динамічних образів операцій присвоювання, порівняння, передачі параметрів в процедури та функції, рекурсивних викликів процедур та функцій, процесу генерації вхідних даних робить середовище демонстрації виключно корисним засобом вивчення основ алгоритмізації». З цього ж приводу Волошинов С.А. [35, с. 36] зазначає, що «при вивченні алгоритмів опрацювання даних, що представляється різними структурами даних, важливу роль грають візуалізатори алгоритмів, що дозволяють в наочній формі динамічно відображати деталі їх роботи». Це пов’язане з тим, що для деяких алгоритмів динамічний варіант демонстрації його роботи є природнішим, ніж набір статичних ілюстрацій. Для споріднених алгоритмів (наприклад, алгоритмів сортування) візуалізація надає можливість наочно продемонструвати як загальний підхід, так і відмінність в механізмах їх дії. Це відкриває можливість використання зазначених технологій при вивченні окремих розділів математики і програмування. Саме ж поняття “візуалізатор” Волошинов С.А визначає, як «... програма, в процесі роботи якої на екрані комп’ютера динамічно демонструється застосування алгоритму до вибраного набору даних. Візуалізатори дозволяють вивчати роботу алгоритмів в покроковому режимі, аналогічному режиму трасування програм. Вони при необхідності допускають трасування укрупненими кроками, ігноруючи рутинну частину обчислювального процесу, що істотно, наприклад, для перебірних алгоритмів».

У роботі Рамського Ю.С. та Цибко Г.Ю. [196, с. 15] подано етапи розв’язання задач за допомогою комп’ютера: «Розв’язання будь-якої практичної задачі містить ряд послідовних етапів: розробка математичної моделі (математична постановка задачі); добір методу розв’язування задачі; побудова алгоритму розв’язування; складання програми (запис алгоритму мовою програмування); введення програми в комп’ютер і перевірка її правильності (налагодження і тестування); виконання програми за допомогою комп’ютера; аналіз отриманих результатів». Як зазначає Т. П. Кобильник [117, с. 10] «Розв’язуючи прикладні

задачі, необхідно вміти абстрагуватись, виокремлювати суттєві і несуттєві характеристики об’єкта, системи».

Навчання програмування передбачає вивчення певної мови програмування та розвиток певної системи мислення, притаманної даному різновиду людської діяльності. Отже, має значення не лише сам факт навчання програмування, а й те, як воно буде здійснюватись, якою мовою програмування студенти будуть оволодівати, яка система мислення буде формуватись у результаті такого навчання. Дейкстра писав: «Найважливішою, але самою непомітною властивістю будь-якого інструменту є вплив його на формування звичок людей, які ним користуються. Коли цей інструмент – мова програмування, його вплив, незалежно від нашого бажання, позначається на нашому способі мислення [66, с. 11].

Кількість різноманітних мов програмування постійно зростає. Поява нових мов програмування та парадигм, на яких вони базуються, – закономірний процес еволюції ІКТ [143; 225]. Серед значної кількості мов, на основі яких може відбуватися процес навчання програмування, існують такі, що не лише широко застосовуються, а й мають подібні характеристики. Поряд із цим, є й специфічні та малоперспективні мови програмування: рідко використовувані, морально застарілі, незручно організовані. Добір мови програмування, що буде використовуватись для написання певної комп’ютерної програми, є надзвичайно важливим етапом розробки програмного забезпечення [113]. Не менш важливим є добір мови програмування, що буде використовуватися для навчання [242].

Варто зважити на те, що добір мови програмування, що буде використовуватися для навчання програмування, певною мірою визначає як ефективність її функціонування ,так і зручність самого процесу навчання. У своїй роботі [242, с. 598] Шевчук П. Г. зазначає, що “добір мови програмування зазвичай визначається трьома важливими умовами: характеристиками, особливостями самої мови; наявністю зручного в навчанні, доступного до використання середовища програмування; наявністю методичної підтримки (інформаційно-дидактичного та навчально-методичного забезпечення)”. Окрім того необхідно відмітити, що найважливішими характеристиками мови

програмування, що використовується для навчання є призначення, підтримувані парадигми, розповсюдженість, алфавіт та особливості синтаксису.

Серед мов програмування, що традиційно використовуються для навчання, можна, досить не строго, за схожістю між собою, виокремити три найбільш поширених різновидності синтаксисів: синтаксис мови Basic, синтаксис мови Pascal, синтаксис мови C [6].

Зважаючи на сказане можна зазначити, що навчання програмування може базуватися на вивченні різних мов програмування. Володіння різними парадигмами програмування, знання кількох мов, розуміння головних відмінних рис мов програмування значно полегшує програмісту засвоєння нових мов програмування і надає можливість крокувати разом з усією індустрією створення програмного забезпечення. Процес навчання може проводитися навіть спираючись на декілька мов програмування одночасно.

Проте частина науковців притримується думки, що навчання програмування необхідно проводити в межах однієї мови програмування, з можливим подальшим вивчення інших мов програмування. В роботі Н.В.Іванової та Лю Минь [101] при вивченні нової мови програмування пропонують використовувати аналогію на основі знань про вже вивчену мову програмування. Щодо вибору мови програмування, на якій необхідно починати вивчення програмування, то у своїй роботі [59] Гришко Л.В. вказує на вибір мови програмування Pascal: «Вибір мови програмування Pascal обумовлений тим, що, по-перше, ця мова спеціально створювалася для навчання програмуванню; по-друге, основи цієї мови, як правило, відомі студентам зі шкільного курсу інформатики, оскільки в більшості регіонів України в шкільному курсі інформатики навчають алгоритмізації і програмуванню саме на прикладі мови програмування Pascal. Тобто пропонується до процесу навчання алгоритмізації і програмування включити пропедевтичний курс, у якому передбачено навчання з використанням мови програмування високого рівня Pascal». У своїй роботі [215, с. 36] Співаковський О.В. розглядає питання методологій програмування, зокрема вказує: «… методологія об’єктно-орієнтованого проектування, безумовно, є передовою і ефективною для

проектування широкого спектру великих програмних систем, таких, як інтерактивні системи, системи реального часу. Концепції ООП добре поєднуються з іншими підходами до написання програмних систем. Методи ООП займуть гідне місце й у технологіях програмування майбутнього, органічно сполучаючись з іншими (як старими, добре відомими, так і новими, ще не усвідомленими і не сформульованими) підходами до проектування великих програмних систем». Про переваги ООП до розробки ПЗ говорить і Горошко Ю.В. у своїй роботі [52, с. 88] «Розробка програмного забезпечення – складний і трудомісткий процес. При традиційному, не об'єктно-орієнтованому підході, відбувається лавиноподібне нарощування складності розробки програми. Подолати ці проблеми можна шляхом правильного використання об'єктно-орієнтованого підходу до програмування».

Співаковський О.В. у своїй роботі [215] торкається і питання вибору середовища програмування, яке тісно пов'язане з мовою програмування. В ній [215, с. 121] він каже: «У процесі розгляду середовищ програмування будемо враховувати засоби розробки Windows-додатків, які мають значне розповсюдження в навчальних закладах. На сучасному етапі найбільш популярними є Delphi, Visual Basic, Visual C++. Ці засоби розробки – це продукти одного класу, забезпечені в цілому достатнім набором засобів і компонентів для створення розвинутих Windows-додатків. Кожен програмний продукт має свої особливості». Далі він проводить аналіз цих середовищ і на його основі робить висновок: «Виходячи з перерахованих міркувань, у першу чергу через популярність, як засіб розробки обираємо Delphi, яке в свою чергу використовує мову програмування ObjectPascal». Варто зазначити, що мова Pascal має декілька діалектів серед яких: Turbo Pascal, Borland Pascal, Pascal ABC, Free Pascal, причому деякі з них є пропрієтарними. На нашу думку, зважаючи на необхідність формування певних етичних та правових норм поведінки стосовно використання засобів комп'ютерних технологій та даних, що зберігаються за їх допомогою, бажано спиратися на вільно розповсюджувані ПЗ. Отже процес навчання програмування може базуватися на вивченні мови Pascal, використовуючи вільно

розповсюджуваний компілятор Free Pascal. А вивчення ООП та навчання створення додатків з графічним інтерфейсом варто проводити в середовищі Lazarus, оскільки воно є вільним середовищем розробки програмного забезпечення для компілятора Free Pascal на мові Object Pascal та розповсюджується під вільними ліцензіями GNU General Public License (GNU GPL) та GNU Lesser General Public License (GNU LGPL). Окрім того Lazarus працює під управлінням ОС Linux, Mac OS X, UNIX-подібних, Windows, Android, в результаті чого додатки, створені в ОС Windows, без великих зусиль можуть бути перекомпільовані під іншу ОС. Тобто, використовуючи середовище Lazarus, можна створювати крос-платформенні додатки.

Важливе місце в процесі навчання програмування має самостійна робота. Так в своїй роботі [226, с. 190] Тищенко С.І. вказує: «Однією з провідних якісних характеристик професійної підготовки майбутніх фахівців з програмування є результативність організації самостійної роботи студентів. Самостійну роботу студентів в процесі навчання програмування визначено як форму навчання, під час якої студент оволодіває компетентностями, навчається планомірно й систематично працювати, мислити, формує свій стиль розумової діяльності. Відмінність її від інших форм навчання полягає в можливості студента самостійно організовувати свою діяльність відповідно до поставлених завдань, тобто становить собою єдність творчості й пізнання». Питанню самостійної роботи присвячена і робота Ільченко А.А. [244, с. 115] в якій вона вказує, що «під час організації самостійної роботи майбутніх фахівців з програмування необхідно подолати суперечності, що мають місце, а саме – між тенденцією збільшення обсягу самостійної роботи студентів згідно принципів Болонського процесу та особливостей її виконання; збільшенням питомої ваги самостійної роботи у процесі навчання циклу професійно-практичних дисциплін і досконалістю технологій самостійної роботи; становленням особистісної орієнтації в навчальному процесі та недостатньою готовністю викладачів і студентів до виконання нових функцій під час здійснення самостійної роботи; рівнями

теоретичної та практичної підготовленості до професійної діяльності майбутніх фахівців з програмування».

Для фахової освіти характерним є розвиток спеціальних здібностей майбутніх фахівців з програмування, важливих для активної кваліфікованої діяльності; розвиток низки потреб і мотивів, пов'язаних із спеціалізацією та кваліфікацією майбутніх фахівців; формування компетентностей майбутніх фахівців з програмування для вдосконалення професійної кваліфікації на основі самоосвіти. Ми вважаємо, що належний рівень компетентностей майбутніх фахівців з програмування забезпечується реалізацією вимог щодо особистісної активної орієнтації сучасної професійної освіти. Так Тищенко С.І. у своїй роботі [226, с. 53] вказує на те, що «… сучасним критерієм ефективної професійної підготовки програміста є його спроможність поповнювати знання та уміння протягом життя відповідно до постійно зростаючих вимог до його компетентносних характеристик; особливої значущості при цьому набуває проблема збереження попередньо здобутих знань та умінь». Як зазначено в роботі [59, с. 23]: «Відповідно до компетентністного підходу у навчанні алгоритмізації і програмування в роботі визначено компетентності студентів комп'ютерних спеціальностей: загальні – наявність здатностей до організації й планування своєї навчальної і професійної діяльності, здобувати й аналізувати відомості з різних джерел, формалізувати і накопичувати здобуті знання, працювати в колективі, використовувати знання іноземних мов; спеціальні – наявність здатностей налагоджувати й тестувати навчальні програми в середовищі системи програмування, застосовувати й комбінувати типові алгоритми й відомі методи програмування, розуміти галузі застосування здобутих знань в інших професійно-орієнтованих і спеціальних дисциплінах та в сфері майбутньої професійної діяльності, до вивчення нових мов програмування».

Закріплення знань студентів та визначення їх навчальних досягнень за вивченими питаннями здійснюється в ході виконання лабораторних робіт. Як зазначає Меджитова Л.М у роботі [157, с. 52]: «Важливо відзначити, що при виконанні лабораторних робіт бажано використовувати диференційований підхід,

який полягає в підготовці завдань різного рівня складності. На перших заняттях при проведенні оцінювальної роботи студентам надається можливість самостійно вибрати бажаний рівень складності завдання. Однак згодом на основі результатів попередніх робіт викладач допомагає у виборі посильного завдання. При необхідності рівень складності завдання для конкретного студента може бути дещо знижений або навпаки підвищений. Виконання кожної оцінювальної роботи і подання її викладачеві відбувається в строго встановлені терміни. При цьому переслідується така мета: студент не повинен відкладати виконання завдання до такого часу, коли він вже повинен буде працювати над наступним». Стосовно лабораторних робіт Морзе Н.В. [160, с. 7] зазначає: «Доцільно проводити, крім практичних робіт, ще й лабораторні, до яких необхідно пропонувати відповідні інструкції. В них теоретичний матеріал містить відомості з відповідної теми, аналіз алгоритмів типових задач і варіант опису програми мовою програмування» та зазначає, що «порядок виконання лабораторної роботи має включати три етапи: 1) виконання готових програм, відповідних розглянутим алгоритмам; 2) внесення виправлень до тексту готової програми або розробка нової; 3) виконання виправленої або новоствореної програми».

Для забезпечення якісного засвоєння студентами знань з програмування потрібно не тільки нарощувати число досліджуваних понять, але і вивчати ті самі поняття на різних рівнях деталізації, наукової строгості. Однією з характеристик професійних якостей програміста є розуміння програм. Розуміння програм необхідне при налагодженні і при модифікації програм, а також при навчанні програмування. Б. Шнейдерман у своїх дослідженнях припускає, що для розуміння програми програміст будує багаторівневу внутрішню семантичну структуру, використовуючи свої знання синтаксису мови програмування. На найвищому рівні програміст повинен розуміти, яка задача розв’язується за допомогою програми. На більш низьких семантичних рівнях програміст може розуміти знайомі набори операторів. Можлива ситуація, коли програміст може зрозуміти деталі нижнього рівня, не розуміючи загальної задачі. «Головне полягає в тому, що програмісти виробляють внутрішню семантичну структуру для

подання синтаксису програми, але не запам’ятовують і не розуміють програму як впорядкований набір рядків» [244, с. 120]. Процес запам’ятовування, за допомогою якого програміст перетворює програму у внутрішню семантичну структуру, має “шматковий” характер [244]. Замість того, щоб аналізувати програму символ за символом, програміст розпізнає призначення груп операторів, потім збирає ці фрагменти разом для формування більш великих фрагментів доти, поки не буде зрозуміла вся програма. Тому, що внутрішня семантична структура програми розробляється самим програмістом, ці знання не швидко забуваються і доступні для різних цілей: використання інших структур даних для розв’язування задачі, написання програми розв’язування задачі іншою мовою програмування, роз’яснення призначення програми кому-небудь.

Важливу ланку у навчанні програмування займає навчання створення ПЗ та ППЗ. Створення сучасних комп’ютерних програм – це не тільки і не стільки розробка окремих алгоритмів, як побудова цілісних інформаційних моделей, які максимально відображають реальні явища чи предмети, що моделюються. Під час розвитку галузі програмування формувались та змінювались підходи до побудови комп’ютерних програм, що зумовлювалось постійним ускладненням самих програм, зростанням складності завдань моделювання предметної галузі. У дослідженні [59, с. 35] було виявлено якості, які властиві особистості програмістів, що пов’язані безпосередньо із створенням програмного продукту, серед яких визначені «здатність визначити архітектуру програми; уміння бачити задачу одночасно на різних рівнях деталізації; уміння застосовувати і комбінувати добре відомі прийоми програмування і типові алгоритми; здатність модифікувати програми; уміння запам’ятовувати і відтворювати текст програм; творчі властивості мислення; наявність комплексного мислення; культура власної праці; уміння працювати в колективі; уміння працювати з користувачем і т.п.». Як зазначають автори статті “Модель системи соціально-професійних компетентностей вчителя інформатики” [86, с. 5] Жалдак М.І., Рамський Ю.С., Рафальська М.В.: «На особливу увагу заслуговує процес підготовки вчителя інформатики, оскільки, за наявного стану інформатизації навчального процесу

саме на нього лягає основне навантаження стосовно розробки та впровадження засобів ІКТ в навчальний процес школи, добору і розробки ППЗ та їх педагогічно виваженого використання у процесі навчання різних навчальних предметів, організації телекомунікаційних проектів, створення умов для формування інформатичних компетентностей учнів».

Питанням, присвяченим розробці ППЗ, займаються Жалдак М.І., Горошко Ю.В., Співаковський О.В., Триус Ю.В., Раков С.А. У своїх роботах Співаковський О.В. вказує «З одного боку, ППЗ – це пакети прикладних програм для використання в процесі навчання різних предметів. З іншого боку – це дидактичні засоби, призначені для досягнення цілей навчання: формування знань, умінь і навичок, контролю якості їх засвоєння тощо, тобто це компоненти процесу навчання». Проте він же зазначає [213, с. 23] «… навчальна практична діяльність має певну специфіку. Зокрема, метою учня є побудова ходу розв'язування математичної задачі, а не лише отримання відповіді. Учитель оцінює лише це. Тому педагогічно орієнтовані математичні системи повинні підтримувати саме процес розв'язування математичної задачі. Відомі педагогічно-орієнтовані системи підтримки математичної діяльності Gran, "Динамічна геометрія" (DG) орієнтовані на такий спосіб використання. Зазначимо, що ці навчальні програми спрямовано на математичні задачі, що мають графічну інтерпретацію. У той же час існує широкий клас математичних задач, у яких основними методами розв'язування є алгебраїчні(символьні) перетворення». Варто зауважити, що деякі ПЗ які пропонуються для використовуються в навчальному процесі спираються на використання лише наочність. І як зазначає у своїй роботі [207, с. 179] Скафа О.І. «Большинство ППС, по сути дела, являются демонстрационными, без учета и понимания дидактических принципов, которые должны быть заложены в основу при их создании» та вказує «… чтобы ППС отвечали современным требованиям, предъявляемым к ним педагогами, необходимо, чтобы они представляли собой некую интеллектуальную систему, используя которую учащиеся будут способны самостоятельно выдвигать гипотезы, делать допущения относительно наблюдаемых закономерностей, иметь возможность

экспериментально их наблюдать». При створенні ПЗ та ППЗ необхідно не забувати і про їх якість, як зазначає Дідковська М.В. у своїй роботі [73, с. 35]: «Гарантування якості ПЗ – це комплексна проблема, розв'язання якої вимагає проведення комплексного дослідження та відповідних ресурсів − розробників, фінансів та часу» та зазначає, що «… про надійність ПЗ починають говорити на завершальних стадіях його розроблення. Це пов'язано з тим, що можливість оцінювати надійність програмного продукту з'являлася тільки після завершення його розробки, тобто, коли накопичувалися реальні дані для проведення статистичного аналізу та отримання відповідних оцінок. Власне кажучи, такий підхід надає можливість оцінити надійність вже створеного ПЗ, починаючи з фази тестування. Проте, якщо отримані оцінки не задовольняють поставленим вимогам, то виникає потреба повертатися на попередні етапи, вносити зміни, які іноді приводять до перепроектування системи чи повторення інших стадій. Наслідком є значне зростання як фінансових, так і часових витрат. Сучасні технології створення ПЗ, побудовані на ідеях ітеративного проектування, концептуального моделювання і автоматизованого програмування, надають можливість враховувати вимоги до ПЗ на кожному етапі життєвого циклу».

1.2. Психолого-педагогічні складові готовності майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів

На основі аналізу науково-педагогічної літератури можна зробити висновок про відсутність єдиного підходу до розуміння феномена готовності. Так, у дослідженні А. Рєзановича готовність трактується «як внутрішня якість особистості, в якій виражено її прихильність до здійснення діяльності, а також ступінь засвоєння нею елементів відповідного соціального досвіду та здатність використовувати цей досвід у професійній діяльності.» [197, с. 44]. У дослідженні с. Бризгалової під готовністю розуміється цілісне особистісне динамічне утворення, придбане у результаті спеціального навчання, яке включає у свою структуру взаємопов'язані елементи: науково – теоретичний, практичний та психологічний [25, с. 32]. А. Ф. Линенко визначає готовність як цілісне

утворення, яке характеризує емотивно-когнітивну і вольову мобілізаційність суб'єкта в момент його залучення в певну діяльність.

В психологічній літературі серед різних підходів до визначення поняття «готовності» можна визначити два підходи [13; 14]: розуміння готовності як певного психологічного стану та розуміння готовності як певної якості чи системи якостей людини. Представником першого підходу є П.П. Горностай, який розглядає готовність особистості як цілісне направлене вираження особистості, що включає систему професійних компетентностей, за рахунок яких отримується можливість успішно включатися в професійну діяльність і здійснювати її педагогічно доцільним для даної діяльності і даної особистості чином. М. І. Дяченко, Л. А. Кандибович визначають готовність як налаштованість особистості, настанову на певну дію. Готовність, на думку авторів, це пристосування можливостей особистості для успішних дій в даний момент, внутрішня налаштованість на певну поведінку при виконанні навчальних і трудових задач, установка на активні і доцільні дії [76, с. 18]. Таку готовність вчені називають тимчасовою. Тривала готовність розглядається ними, як система професійно важливих якостей особистості: її досвід, знання, вміння, що необхідні для успішної роботи [76, с. 20]. с. Л. Рубинштейн, як і інші представники другого підходу, розглядає готовність, як якості особистості, що зумовлюють її суспільно значиму поведінку і поряд з системою мотивів та задач включають здатності людини виконати ту чи іншу корисну діяльність [198, с. 119].

Перш за все необхідно торкнутися питання формування готовності до професійної діяльності загалом. Так проблема формування готовності студентів до майбутньої професійної діяльності акумулює проблеми психологічної науки, пов'язані із особливостями особистості, рисами її характеру, потенційними можливостями, які обумовлюють успішність професійної підготовки. Така готовність розглядається в безпосередньому зв'язку з формуванням, розвитком і вдосконаленням психічних процесів, станів, якостей особистості, необхідних для успішної діяльності. Можна також зазначити, що готовність, як передумова будь-якої діяльності, є водночас і її результатом.

Дослідники Г. О. Балл [13], П. с. Перепелиця [177] вказують на те, що основу формування готовності до професійної діяльності слід вбачати не в розвитку операційно-технічних умінь та навичок, а в опорі на такий визначальний параметр готовності, як "комплексна здатність".

Якщо ж розглядати готовність майбутнього педагога до професійної діяльності не може обмежуватися характеристиками досвідченості, майстерності, продуктивності праці та її якості. Не менш важливо при оцінці такої готовності визначити внутрішні сили особистості, її потенціали та резерви, які є суттєвими для підвищення продуктивності професійної діяльності вчителя в майбутньому.

А.Ф. Линенко у своїй роботі [146] пропонує для студентів педагогічних ВНЗ такі елементи готовності до професійної діяльності: професійна самосвідомість, ставлення до діяльності педагога, мотиви діяльності, знання про предмет і способи діяльності, навички і вміння їх практичного втілення, а також професійно значимі риси особистості. В свою чергу с. Н. Абдувахідов елементами професійної готовності майбутніх учителів до педагогічної діяльності називає: ступінь розуміння соціального змісту праці вчителя; активно-позитивне ставлення до вивчення дисциплін, передбачених навчальним планом; настанову на вдосконалення своїх професійно-педагогічних рис; високі результати навчально-виховної роботи з учнями в період педагогічної практики [2, с. 132].

Також досить важливе значення для майбутніх учителів має психічна та психологічна готовність до педагогічної діяльності. Так А. А. Деркач, визначає елементи психічної та психологічної готовності майбутніх учителів до педагогічної діяльності: прагнення працювати краще, виявляти творчість при проведенні уроків, бути впевненим у своїх педагогічних здібностях, вміти тримати під контролем рівень власного емоційного збудження [70, с. 142].

Далі розглянемо різні структури готовності в залежності від її виду.

Стосовно професійної готовності фахівця, загалом, В. О. сластьоніна визначає такі компоненти [208]:

- психологічний – сформованість певного ступеня спрямованості на професійну діяльність;

- науково-методичний – передбачає володіння повним обсягом суспільно-політичних, психолого-педагогічних і спеціальних знань, необхідних для професійної діяльності;
- практичний – наявність сформованих на прогнозованому рівні професійних умінь і навичок;
- психофізіологічний – наявність відповідних передумов для професійної діяльності і оволодіння певною спеціальністю;
- фізичний – відповідність стану здоров'я та фізичного розвитку вимогам професійної діяльності і професійної працездатності.

Якщо ж розглядати готовність до педагогічної діяльності майбутнього вчителя то у своїй роботі український педагог О. М. Коберник визначає такі основні компоненти [114, с. 107]:

- мотиваційний – установка на особливу значущість і важливість нових освітніх технологій у сучасному навчально-виховному процесі; прагнення до активного вивчення педагогічних інновацій; бажання майбутнього вчителя творчо й не ординарно проектувати педагогічну діяльність;
- когнітивний – інтеграція психологічних, педагогічних і технологічних знань;
- операційний – уміння як такі інтегровані якості, що сформовані у студентів під час опанування змісту психолого-педагогічних і фахових дисциплін, самостійної діяльності в період педагогічних практик.

Проведений аналіз літератури з проблем готовності майбутніх учителів до професійної педагогічної діяльності надає можливість визначити основні підходи до визначення такої готовності:

- на особистісному рівні готовність розглядається, як багатопланова структура якостей, властивостей та станів, які в сукупності надають можливість більш-менш успішно здійснювати діяльність;
- на функціональному рівні готовність – це результат підготовки до певної діяльності; деяке інтегративно-особистісне утворення, що включає різні

компоненти: сукупність вмінь, навичок, особистісних якостей, адекватних вимог і змістовну діяльність.

На основі цього розглянемо психолого-педагогічні складові готовності майбутніх учителів математики та інформатикидо розробки ППЗ.

Підготовка студентів педагогічних ВНЗ до розробки ППЗ, результатом якої є готовність до даного виду професійної діяльності, здійснюється в процесі загальної професійної підготовки майбутнього вчителя і має спільні з нею компоненти. В той самий час вона має власні специфічні особливості, обумовлені характером навчальної діяльності і вимогами до особистості, що її здійснює. Узагальнюючи теоретичні положення, запропоновані різними дослідниками, можна зробити висновок про те, що структура готовності майбутніх учителів до розробки ППЗ, як складна динамічна структура, яка наповнена якісними характеристиками та показниками, охоплює такі компоненти:

- мотиваційний компонент передбачає усвідомлене відношення майбутнього педагога до необхідності розробки ППЗ і їх ролі в розв’язанні актуальних проблем сучасної освіти;
- когнітивний компонент включає теоретичні знання про науково-методичні підходи використання ППЗ в навчальному процесі; основні поняття технологій та підходів створення ППЗ; сутність, специфіку та види ППЗ;
- діяльністний компонент передбачає наявність уміння спілкуватися з використанням інформаційних засобів і технологій; уміння не лише визначати основні компоненти та необхідні об’єкти майбутнього ППЗ, а і писати його код; уміння працювати з середовищами розробки ПЗ; уміння приймати ефективні рішення в проблемних ситуаціях;
- ціннісно-рефлексивний компонент включає самоаналіз і самооцінку професійної діяльності на основі інформаційних технологій; здатність до рефлексії у сфері пошуку та перетворення даних, в опануванні та використанні технологій створення ППЗ; наявність власної позиції щодо підходів створення та використання ППЗ у навчально-пізнавальній

діяльності для розв’язання різноманітних задач; прагнення до самоактуалізації, саморозвитку;

- емоційно-вольовий компонент передбачає цілеспрямованість дій в інформаційному середовищі.

Рівень мотивації безпосередньо пов’язаний з когнітивним компонентом професійної діяльності майбутнього вчителя, який на рівні з мотиваційним компонентом входить в керуючу частину дій, пов’язаних з навчальною діяльністю. Ця складова представляє собою результат пізнавальної діяльності і характеризується об’ємом знань, стилем мислення, а в цілому є орієнтованою основою діяльності майбутнього вчителя. Знання збагачують бачення проблематики в області створення ППЗ, виступають необхідною умовою виникнення задач в відповідності з особистісними інтересами та професійними потребами. Педагогічні знання майбутнього вчителя, орієнтованого на розробку та використання ППЗ, можна представити як відомості про методологічні основи інформаційних технологій, їх сутності, характерні показники та різноманітні підходи до класифікації. Знання розглядаються як основа для орієнтації особистості в різноманітті інформаційних технологій, що є передумовою їх виваженого застосування.

Аналіз когнітивного компоненту розглядуваної моделі готовності майбутнього вчителя до розробки ППЗ в якості критеріального показника надає можливість виокремити рівень інформованості про особливості створення ППЗ. Зміст мотиваційного і когнітивного компонентів визначає стратегію професійної поведінки майбутнього вчителя, орієнтованого на створення та використання ППЗ.

Досить важливе місце діяльнісного компоненту готовності майбутнього вчителя до розробки ППЗ займає вміння до певної діяльності, що передбачає свідоме оволодіння цією діяльністю. “Правильно сформовані вміння, – стверджує Є.Н. Кабанова-Меллер, – засновані на знанні способу дії” [105]. Н.В. Кузьмін визначає п’ять інваріантних груп професійних вмінь в структурі педагогічної

діяльності [139]: гностичні, проектувальні, конструктивні, організаційні і комунікативні. Охарактеризуємо їх з позиції розробки ППЗ як діяльності.

Гностичні вміння виражаються у вмінні добувати, поповнювати та розширювати власні знання. Ступінь сформованості гностичних вмінь з точки зору процесу створення ППЗ виражається у вмінні систематично поповнювати та розширювати знання про сучасні технології програмування, що можуть бути використані при розробці ППЗ, шляхом самоосвіти, вивчення досвіду колег, аналізу реальних ППЗ.

Проектувальні вміння виражаються в здатності планувати структуру майбутнього ППЗ у відповідності з предметною галуззю та цілями навчання, психологічними та фізіологічними особливостями учнів, методами навчання, що будуть використовуватися при застосування створеного ППЗ в навчальному процесі.

Конструктивні вміння виражаються в виборі доцільних прийомів і способів розробки ППЗ, доборі і дозуванні необхідних теоретичних даних, виборі найбільш ефективних алгоритмів для реалізації запланованих характеристик.

Організаційні вміння виражаються в здатності організувати власну діяльність і діяльність колективу розробників в відповідності з цілями процесу розробки ППЗ, раціонально розподіляти час та контролювати темп розробки.

Комунікативні вміння виражаються в вмінні використовувати різноманітні механізми формування міжособистісних відносин в колективі розробників ППЗ.

Досить важливим компонентом в структурі готовності є ціннісно-рефлексивний компонент, який характеризує ступінь пізнання навчального матеріалу та аналіз науково-педагогічних підходів майбутнього вчителя в процесі розробки ППЗ.

На основі взаємодії з іншими людьми, коли людина намагається зрозуміти думки та дії іншої людини, вона стає здатною рефлексивно відноситися і до самої себе. Отже, пошук, освоєння і застосування вже відомих технологій створення ППЗ, аналіз отримуваних результатів і особистого стиля проектування ППЗ

можуть призвести до створення нових підходів до проектування та створення нових ППЗ.

Для забезпечення формування готовності майбутнього вчителя до розробки ППЗ необхідні організаційні, технологічні, методичні та педагогічні умови, а саме: створення педагогічної системи, спрямованої на формування і розвиток у майбутнього вчителя первинної психолого-педагогічної культури та основних професійних компетентностей; забезпечення узгодженої та послідовної роботи викладачів різних навчальних дисциплін щодо формування професійної готовності вчителя; використання методичної системи навчання, орієнтованої на цілеспрямоване формування професійних компетентностей майбутнього викладача; пошук нових методів, методик, технологій, засобів та організаційних форм навчання, які б сприяли виявленню потенційних можливостей до професійної діяльності та стимулювали їх розвиток і саморозвиток у процесі самовдосконалення; забезпечення необхідною навчальною, методичною літературою та документацією, що надає можливість ефективно забезпечувати формування готовності майбутніх учителів, сприяє їх самопізнанню та самооцінюванню, формує та вдосконалює систему психологічних, педагогічних та методичних знань, умінь і навичок; упровадження в практику проведення навчальних занять, соціально-психологічних тренінгів та їх окремих елементів, спрямованих на пізнання та сприйняття майбутнім вчителем своїх професійно-важливих рис і на їх цілеспрямований розвиток; активне стимулювання творчості науково-педагогічних працівників у педагогічній діяльності та сприяння прояву їхньої індивідуальності і творчості у цій діяльності.

1.3. Концептуальні засади розроблення програмних засобів

1.3.1. Парадигма програмування. Розглядаючи термін “парадигма” необхідно відмітити, що він є багатозначним і використовується не лише в програмуванні. Так у словнику Даля поняття парадигма визначається, як: “набір теорій, стандартів та методів, які спільно представляють собою спосіб організації наукового знання. Або іншими словами, спосіб бачення світу” [64]. В програмуванні значення цього терміну не дуже відрізняється. Проте багато різних

авторів уточнюють даний термін в межах програмування і дають визначення поняттю “парадигма програмування”, які можуть відрізнятися один від одного.

Термін “парадигма програмування” був використаний Робертом Флойдом, лауреатом премії Тюрінга 1979 року, в лекції «Парадигми програмування» [143, с. 159-174]. Парадигми в програмуванні, на його думку – це спосіб концептуалізації, який визначає, як проводити обчислення і як робота, виконувана комп’ютером, повинна бути структурована і організована [143, с. 163]. Окрім того він каже: “Якщо прогрес мистецтва програмування в цілому потребує постійного вдосконалення парадигм, то вдосконалення мистецтва окремого програміста потребує, щоб він розширював свій репертуар парадигм” [143, с. 168]. Таким чином, на думку Роберта Флойда, на відмінність від парадигми в науковому світі, парадигми програмування можуть поєднуватися, збагачуючи інструментарій програміста.

Розглянемо деякі з інших означень “парадигми програмування”. У своїй роботі [245, с. 140] Деніел Бобров визначає парадигму як: «стиль програмування, як опис намірів програміста». Брюс Драйвер та Лінда Фрідман [254; 248] визначають даний термін досить схоже, а саме “модель чи підхід до розв’язання проблеми” та “підхід до розв’язання проблем програмування”. В свою чергу Пітер Вегнер [264, с. 273] поняття парадигми відносить до мови програмування і зазначає, що парадигма програмування є “правилом класифікації мов програмування в відповідності з деякими умовами, які можуть бути перевірені ”. Діомідіс спінелліс дає таке означення: «Слово парадигма використовується в програмуванні для визначення сімейства позначень (нотацій), які розділяють загальний спосіб (методику) реалізації програми» [262].

Виходячи з вищесказаного поняття парадигми програмування можна сформулювати наступним чином. Парадигми програмування – це сукупність підходів, що визначають стиль написання програми, а також спосіб організації виконання комп’ютером програми.

Більшість сучасних парадигм програмування мають в своїй основі певні принципи та концепції. За своїм походженням одні парадигми якимось чином

пов’язані з іншими: базуючись на них, продовжуючи їх ідеї, чи навпаки, використовуючи радикально протилежні принципи.

Відносно найменування парадигм програмування, різні автори визначають різні терміни. Проте, поки що, не існує єдиної системи найменувань парадигм програмування та їх чіткої класифікації. Назви окремих парадигм мають декілька синонімів або розглядаються в складі інших, більш загальних підходів до написання програмного забезпечення. Проте основними парадигмами програмування можна вважати парадигми імперативного, декларативного та об’єктно-орієнтованого програмування.

Імперативне програмування – це парадигма програмування, за якою процес обчислення описується в вигляді інструкцій, які змінюють стан програми. Тобто це послідовність команд, які має виконати комп’ютер. Імперативне програмування є відбиттям архітектури традиційних ЕОМ, що була запропонована фон Нейманом в 40-х роках 20-го століття. Теоретичною моделлю імперативного програмування служить алгоритмічна система за назвою «машина Тьюринга».

Програма імперативною мовою програмування складається з послідовності операторів (інструкцій), що задають процес розв’язання завдання. В даному випадку основним є оператор присвоювання, який слугує для зміни вмісту областей пам’яті. Концепція пам’яті як сховища значень, вміст якого може обновлятися операторами програми, є фундаментальною в імперативному програмуванні. В результаті цього виконання програми зводиться до послідовного виконання операторів з метою перетворення початкового стану пам’яті (значень вхідних даних) в результати (вихідні дані). Таким чином, з погляду програміста наявні програма й пам’ять, причому перша послідовно оновлює вміст останньої.

Свого подальшого розвитку імперативне програмування отримало в процедурному програмуванні – парадигма якого заснована на концепції виклику процедур (підпрограм, методів, функцій).

Процедурні мови характеризуються наступними особливостями: необхідністю явного керування пам’яттю, зокрема, описом змінних; малою придатністю для символьних обчислень; відсутністю строгої математичної основи; високою ефективністю реалізації.

Одним з найважливіших класифікаційних ознак процедурної мови є її рівень. Рівень мови програмування визначається семантичною (змістовною) ємністю її конструкцій і ступенем її орієнтації на програміста. Чим більше мова орієнтована на людину, тим вище її рівень. Дамо коротку характеристику реалізованим на ЕОМ мовам програмування в порядку зростання їхнього рівня.

Мова асемблер – це система запису програми з деталізацією до окремої машинної команди в якій може використовуватися мнемонічне позначення машинних команд і символічне задання адрес. Використання мови асемблер надає програмістові можливість користуватися мнемонічними кодами операцій, присвоювати зручні імена областям пам’яті, а також задавати найбільш зручні схеми адресації. свого розширення мова асемблер отримала в мові макроасемблера шляхом включення в неї макрозасобів. За допомогою макрозасобів в програмі можна описувати послідовності інструкцій з параметрами – макровизначення. Після цього програміст може використовувати макрокоманди, які в процесі асемблювання програми автоматично заміщаються звичайними командами. Інакше кажучи, мова макроасемблер представляє засоби визначення й використання нових команд як послідовностей базових інструкцій, що підвищує її рівень.

Мови асемблер та макроасемблер застосовуються системними програмістами з метою використання всіх характеристик апаратної складової ЕОМ і одержання ефективної за часом виконання й затратам пам’яті програми. На цих мовах зазвичай розробляються відносно невеликі програми, що входять до складу системного програмного забезпечення (драйвери, утиліти й інші) та окремі підпрограми в межах інших мов програмування.

Мова програмування С спочатку була розроблена для реалізації ОС UNIX на початку 70-х років. Надалі набула високу популярність серед системних і прикладних програмістів.

У мові С поєднуються переваги сучасних високорівневих мов зокрема базових конструкцій алгоритмів і структур даних з можливостями доступу до апаратних засобів ЕОМ на рівні, що звичайно асоціюється з мовою низького рівня типу мови асемблер. Мова С має синтаксис, що забезпечує стислість програми, а компілятори здатні генерувати ефективний код.

Одна з найбільш істотних особливостей мови С полягає у нівелюванні розходжень між виразами й операторами, що наближає її до функційних мов. Зокрема, вираз може мати побічний ефект присвоювання, а також може використовуватися як оператор. Немає також чіткої границі між процедурами й функціями, більше того, поняття процедури не вводиться взагалі. Відсутня і строга типізація даних, що надає додаткові можливості програмістові, але не сприяє написанню надійних програм і на початкових етапах може погіршувати розуміння використовуваних змінних. свого подальшого розвитку отримала в мові С++.

Мова програмування Basic (Beginners All-purpose Symbolic Instruction Code) – багатоцільова мова символічних інструкцій для початківців представляє собою просту мову програмування, розроблену в 1963 році для використання початківцями. Вона була розроблена як найпростіша мова для безпосереднього спілкування людини з обчислювальною машиною. Тому спочатку робота велася в інтерактивному режимі з використанням інтерпретаторів.

Сучасний Basic відійшов від початкових концепцій і тепер представляє типову мову програмування зі строгою типізацією та іншими атрибутами високорівневих мов програмування. На даний час існує значна кількість реалізацій даної мови програмування, зокрема: Visual Basic for Applications, Visual Basic .NET, VBScript.

Мова програмування Pascal є досить популярною процедурною мовою програмування. Розроблена в 1970 році швейцарським фахівцем в області

обчислювальної техніки професором Н. Віртом, мова названа на честь французького математика й за задумом автора призначалася для навчання програмування. Однак мова вийшла настільки вдалою, що стала важливим інструментом при розв'язанні задач обчислювального й інформаційно-логічного характеру. В 1979 році був підготовлений проект опису мови – Британський стандарт мови програмування PascalBS6192, що став також і міжнародним стандартом ISO7185.

У мові Pascal реалізовано ряд концепцій, які розглядаються як основа «дисциплінованого» програмування й запозичених згодом розробниками багатьох мов. Однією з істотних ознак мови Pascal є послідовна й досить повна реалізація концепції структурного програмування. Причому це здійснюється не тільки шляхом упорядкування зв'язків між фрагментами програми та керуванням, але й за допомогою структуризації даних. Крім того, у мові реалізована концепція визначення нових типів даних на основі вже наявних. Ця мова, на відміну від мови С, є строго типізованою. Мова Pascal характеризується: високим рівнем; широкими можливостями; стрункістю, простотою й стислістю; строгістю, що сприяє написанню ефективних і надійних програм; високою ефективністю реалізації на ЕОМ.

На даний час широко використовуються такі версії цієї мови, як ObjectPascal та FreePascal.

Декларативне програмування – це парадигма програмування, відповідно до якої, програміст описує, який результат необхідно отримати, замість описання послідовності команд для отримання цього результату [67]. Парадигма декларативного програмування широко використовується в системах штучного інтелекту.

З однієї сторони програма є декларативною, якщо вона містить опис, що має бути, а не містить інструкції, як це зробити. Проте з іншої сторони програма є декларативною, якщо вона написана базуючись на логічному, функційному програмуванні чи програмуванні на обмеженнях.

Центральним поняттям у декларативному програмуванні є відношення. Програма, заснована на декларативному програмуванні, представляє собою сукупність визначених відносин між об’єктами (у термінах умов або обмежень) і мети (запиту). В декларативному програмуванні потрібно тільки специфікувати факти, на яких ґрунтується алгоритм, а не визначати послідовність кроків, які потрібно виконати. Процес виконання програми трактується як процес пошуку логічної формули, побудованої за правилами згідно встановлених відносин. Результат обчислення є побічним продуктом цього процесу.

Мови декларативного програмування характеризуються: високим рівнем; орієнтацією на символьні обчислення; можливістю інверсних обчислень, тобто змінні в процедурах не діляться на вхідні й вихідні;

Декларативні програми, як правило, мають невелику швидкодію, тому що обчислення здійснюються методом проб і помилок, пошуком з поверненнями до попередніх кроків.

Розберемо детальніше парадигми програмування, які є складовими декларативного програмування.

Логічне програмування – це парадигма програмування, за якою виведення нових фактів відбувається з даних фактів та за заданими логічними правилами. Логічні мови програмування базуються на теорії і апараті математичної логіки з використанням математичних принципів резолюції. самою відомою мовою логічного програмування є мова Prolog (PROgrammingin LOGic – програмування в термінах логіки). Ця мова була створена французьким вченим А. Кольмерое в 1973 році, і має понад 15 різних її реалізації. На даний час відомі й інші мови, зокрема Visual Prolog, Oz, Mercury.

Функційне програмування – це парадигма програмування, за якою процес обчислення трактується як обчислення значень функцій в математичному розумінні останніх (на відмінність від функцій як підпрограм в процедурному програмуванні). На практиці відмінність математичної функції від поняття «функції» в функційному програмуванні полягає в тому, що дані функції взаємодіють і змінюють вже визначені дані. Таким чином, в функційному

програмуванні, при виклику однієї і тієї ж функції з однаковими параметрами можна отримати різні дані на виході, із-за впливу на функцію зовнішніх факторів. А в імперативному програмуванні при виклику функції з одними і тими ж аргументами завжди отримається однаковий результат [237].

Сутність функційного програмування А.П. Єршов визначає як «... спосіб складання програм, у яких єдиною дією є виклик функції, єдиним способом розчленовування програми на частини є введення імені для функції, а єдиним правилом композиції є оператор суперпозиції функції. Ніяких комірок пам'яті, ні операторів присвоювання, ні циклів, ні, тим більше, блок-схем, ні передачі керування» [78, с. 350].

Першою функційною мовою можна вважати мову LISP (LIStProcessing – опрацювання списків), створену в 1959 році. Мета її створення полягала в організації зручного опрацювання символьних даних. Головна риса цієї мови – уніфікація програмних структур і структур даних. серед інших мов функційного програмування можна виокремити Háskell, Miranda , Hope, Erlang, F#.

Програмування в обмеженнях – це парадигма програмування, у якій відношення між змінними зазначені у формі обмежень. Обмеження відрізняються від загальних примітивів мов імперативного програмування тим, що вони визначають не послідовність кроків для виконання, а властивості шуканого розв'язку [187].

Серед мов з обмеженнями можна виокремити: Prolog III, Prolog IV.

Об'єктно-орієнтоване програмування – це парадигма програмування за якою основними концепціями є поняття класів і об'єктів, які взаємодіють між собою за допомогою повідомлень.

Прототипом об'єктно-орієнтованого програмування послужив ряд засобів, що входили до складу мови Simula-67. Але в самостійний стиль ООП оформилося з появою мови SmallTalk, розробленої А. Кеєм в 1972 році і призначеної для реалізації функцій машинної графіки.

Як зазначалося, в основі об'єктно-орієнтованого програмування лежить поняття об'єкта, а суть його виражається формулою: «об'єкт = дані + процедури».

Кожний об’єкт інтегрує в собі деяку структуру даних і доступні тільки йому процедури опрацювання цих даних.

Для опису об’єктів служать класи. Клас визначає властивості й методи об’єкта, що належить цьому класу. Відповідно, будь-який об’єкт можна визначити як екземпляр певного класу.

ООП полягає у виборі наявних або створенні нових об’єктів і організації взаємодії між ними. При створенні нових об’єктів властивості об’єктів можуть додаватися або успадковуватися від об’єктів-предків. У процесі роботи з об’єктами допускається поліморфізм, який полягає в тому, що одне і те саме повідомлення, будучи відісланим різним об’єктам, може призвести до виконання різних дій (виклику різних методів) в залежності від того, який конкретний об’єкт на етапі виконання програми є отримувачем цього повідомлення.

До сучасних об’єктно-орієнтованих мов програмування відносяться: C++, Java, Object Pascal, Python, C#, Ruby. Класичними прикладами об’єктно-орієнтованих мов програмування залишаються C++, Java, Object Pascal.

Мова C++ була розроблена на початку 80-х років 20-го століття Б. страуструпом. Ним була створена компактна система, в якій за основу була взята мова C, доповнена елементами мов BCPL, Simula-67 і Algol-68.

Мова Java є об’єктно-орієнтованою і архітектурно-нейтральною мовою інтерпретуючого типу, що забезпечує надійність, безпеку, мобільність, високу продуктивність в сполученні із багатопоточністю та динамічністю.

Object Pascal − мова програмування, розробка якої почалася в 1986 році з більш ранньої версії мови, яка мала назву Clascal. В мові Object Pascal використовується класичний синтаксис мови Pascal, що забезпечує її легке використання в навчальному процесі і більш плавний перехід від імперативного до об’єктно-орієнтованого програмування.

З погляду характеристик, властивих об’єктно-орієнтованим мовам, Java володіє рядом переваг перед мовами C++ та Object Pascal. Так, мова Java є більш гнучкою й потужною системою інкапсуляції даних. Механізм спадкування, реалізований в Java, зобов’язує до більш строгого підходу, що поліпшує

надійність і розуміння коду. Принциповою різницею між мовами є те, що Java є інтерпретовною, а C++ та Object Pascal – компільованими.

На сучасному етапі в силу своєї конструктивності ідеї об'єктно-орієнтованого програмування використовуються в багатьох процедурних мовах.

Останнім часом багато програм, особливо тих, що базуються на об'єктно-орієнтованих мовах, створюються у середовищах візуального програмування. Характерною рисою таких середовище є розробка програм з готових «будівельних блоків», що надає можливість розробнику створити інтерфейсну частину програмного продукту в візуальному режимі, практично без кодування програмних операцій. До числа об'єктно-орієнтованих середовищ візуального програмування належать VisualStudio, Delphi, Lazarus, C++ Builder, Eclipse.

Більшість сучасних об'єктно-орієнтованих мов програмування спираються на окремі парадигми, які є похідними від об'єктно-орієнтованої парадигми. серед таких парадигм можна виокремити компонентно-орієнтоване та подіє-орієнтоване програмування.

Компонентно-орієнтоване програмування – це парадигма програмування, ключовим елементом якої є компонент (об'єкт класу), який компілюється незалежно від інших, а на стадії виконання необхідні компоненти під'єднуються динамічно. До сучасних компонентно-орієнтованих мов програмування відносяться більшість об'єктно-орієнтованих мов програмування для ких наявне середовище візуального програмування.

Подіє-орієнтоване програмування – це парадигма програмування, в якій програма складається з окремих об'єктів, які реагують на події. Подію може викликати дія користувача (наприклад, натиснення кнопки в додатку), операційна система, або подію можна створити у самій програмі. Всі події відслідковуються ОС, за якою на кожен тип події формується відповідне повідомлення. Це повідомлення передається в програму, яка реагує на неї у відповідності з алгоритмом своєї роботи. Роль програміста полягає в розробці коду по опрацюванню повідомлень при виникненні подій. До сучасних подіє-

орієнтованих мов програмування відносяться: Perl Object Environment, Prado (інструмент для Web-програмування на PHP 5).

Таким чином в загальному розумінні парадигма програмування визначає те, в яких термінах програміст описує логіку програми. Наприклад, в імперативному програмуванні програма описується, як послідовність дій, а в декларативному програмуванні єдиною дією є виклик запиту. В об'єктно-орієнтованому програмуванні програму прийнято розглядати як набір об'єктів, що взаємодіють. ООП можна вважати поєднанням імперативного та декларативного програмування, яке доповнене принципом інкапсуляції даних і методів в об'єкт.

Також необхідно зазначити, що парадигма програмування не визначає однозначно мову програмування. Багато сучасних мов програмування є мультипарадигменними, тобто допускають використання різних парадигм. Так мовою c++, яка є об'єктно-орієнтованою, можна писати чисто імперативні програми без використання об'єктів, а на Ruby, в основу якого в значній мірі покладена об'єктно-орієнтовна парадигма, можна писати програми використовуючи принципи функціонального програмування.

1.3.2. Підходи до проектування ПЗ. Процес проектування програми починається після складання вимог щодо її функціонування й закінчується перед реалізацією, кодуванням програми на деякій конкретній мові. Ретельне виконання процесу проектування покликано підвищити практично всі показники якості програми, полегшити процес впровадження та супроводу ПЗ. На етапі проектування вирішується питання, як саме ПЗ буде задовольняти встановленим до неї вимогам.

Завданням етапу проектування є дослідження структури ПЗ й логічних взаємозв'язків його елементів, причому тут не розглядаються питання, пов'язані з реалізацією на конкретній платформі. Проектування визначається як «ітераційний процес отримання логічної моделі системи разом зі строго сформульованими цілями, поставленими перед нею, а також написання специфікацій реальної системи, що задовольняє цим вимогам» [30].

В результаті діяльності на етапі проектування повинен бути отриманий проект ПЗ, що містить достатньо даних для реалізації цього ПЗ на його основах у рамках визначених ресурсів та часу.

За напрямом розробки найпоширенішими є такі підходи до процесу проектування: класичне висхідне проектування та класичне низхідне проектування. Під час розгляду підходів до процесу проектування буде використаний термін «модуль», проте дещо в іншому сенсі, ніж в мовах програмування. Поняття модуля в даному аспекті потрібно асоціювати з поняттям підпрограми чи навіть сукупності підпрограм.

Класичне висхідне (знизу нагору або синтетичне) проектування та розробка. В основі цього підходу до проектування програм лежить ідея виокремлення окремих модулів, що реалізують певні функції в загальній програмі. Вибір модулів може визначатися різними міркуваннями: зрозумілістю реалізованих функцій, розмірами, структурами даними, наявністю подібності із уже наявними програмами й можливістю їх зміни для нових цілей.

Спочатку будується модульна структура програми у вигляді дерева. Потім по черзі, починаючи з модулів самого нижнього рівня (листи дерева структури програми) дається опис окремих модулів, визначається інтерфейс вхідних і вихідних даних, потім модуль програмується, налагоджується й тестується. Проте це має відбуватися в такому порядку, щоб для кожного модуля, що програмується, вже були запрограмовані всі модулі, до яких він може звертатися.

Після цього здійснюється об’єднання модулів у підсистеми на основі визначених інтерфейсів і окремо розроблюється керуюча програма, що визначає послідовність викликів цих модулів, доповнених підпрограмами введення-виведення, контролю даних і т.д.. Потім ці підсистеми піддаються налагодженню, в процесі якого поряд з додатковою перевіркою модулів перевіряється їх відповідність заданим інтерфейсам. Далі отримані підсистеми поєднуються в загальний остаточний програмний засіб, який налагоджується та тестується, тобто відбувається остаточна перевірка та виправлення виявлених помилок.

На перший погляд такий порядок розробки програми здається цілком природним: кожний модуль при програмуванні виражається через уже запрограмовані підлеглі модулі, а при тестуванні використовує вже налагоджені модулі. Проте існує ряд недоліків в такому проектуванні [3; 22; 100].

По-перше, для програмування окремого модуля зовсім не потрібно текстів модулів, що ним використовуються – для цього досить, щоб кожний модуль, що використовується, був лише специфікований (в обсязі, що надає можливість побудувати правильне звернення до нього), а для тестування його можливо і навіть корисно модулі, що використовуються, заміняти імітаторами (заглушками). Як імітатор можна використовувати модуль, який представляється досить простим програмним фрагментом. Проте імітатор не просто має виконувати повернення повідомлення про під'єднання модулю, оскільки, коли відбувається виклик деякого модуля, припускається, що він має виконати певну роботу, тобто повернути результат своєї роботи (наприклад, в формі значень вихідних параметрів). В імітаторі також має бути перевірка вхідних параметрів з поверненням, якщо це необхідно, заздалегідь підготовленого коректного результату. Це забезпечуватиме трасування програми.

По-друге, кожна програма в певній мірі підпорядковується деяким внутрішнім для неї, але глобальним для її модулів умовам (принципам реалізації, структурам даних і т.п.), що визначає її концептуальну цілісність і формується в процесі її розробки. При висхідній розробці ці глобальні дані для модулів нижніх рівнів ще не відомі в повному обсязі, тому доводиться їх перепрограмувати, коли при програмуванні інших модулів визначається істотне уточнення цих глобальних даних (наприклад, змінюється глобальна структура даних).

По-третє, при висхідному тестуванні для кожного модуля (крім головного) доводиться створювати власну програму для тестування, що повинна підготувати для модуля, що тестується, необхідний стан інформаційного середовища й зробити необхідне звертання до нього. В той самий час не можна дати ніякої гарантії, що тестування модулів відбувалося саме в тих умовах, у яких вони будуть використовуватися в робочій програмі.

По-четверте, недоліком такого підходу до проектування є складність процесу об’єднання окремих модулів у єдину систему, непередбачуваність цього процесу й труднощі виправлення помилок, допущених на ранніх стадіях розробки.

Варто зазначити, на практиці розробник досить часто використовує вже готові бібліотеки підпрограм для реалізації тих чи інших характеристик власного ПЗ. Тому застосування даного підходу можна охарактеризувати наступним чином: розробник орієнтується в наявних бібліотеках підпрограм і на цих готових бібліотеках намагається сконструювати ПЗ.

Класичне низхідне (зверху вниз або аналітичне) проектування та розробка. Низхідне проектування засноване на ідеї введення рівнів абстракції при аналізі завдання, що розв’язується, і їхньому відображенні на рівні модулів в ієрархічній структурі програми.

Як і в попередньому підході спочатку будується модульна структура програми у вигляді дерева. Потім по черзі програмуються модулі програми, починаючи з модуля самого верхнього рівня (головного), переходячи до програмування іншого модуля. Після того, як всі модулі програми запрограмовані, відбувається їхнє почергове тестування й налагодження в такому ж, низхідному порядку. При такому порядку розробки програми всі необхідні глобальні дані формуються вчасно, тобто ліквідується досить неприємне джерело прорахунків при програмуванні модулів. Істотно полегшується й тестування модулів. Першим тестується головний модуль програми, що представляє всю програму для тестування і тому тестується при "природному" стані інформаційного середовища, при якому починає виконуватися ця програма. При цьому всі модулі, до яких може звертатися головний, заміняються на їхні імітатори.

Після завершення тестування й налагодження деякого модуля відбувається перехід до тестування одного з модулів, які в цей момент були представлені імітаторами, якщо такі є. Для цього імітатор обраного для тестування модуля заміняється на сам цей модуль і додаються імітатори тих модулів, до яких може звертатися обраний для тестування модуль. При цьому кожний такий модуль буде

тестуватися також при "природних" станах інформаційного середовища, що виникають до моменту звернення до цього модуля при виконанні програми для тестування. У такий спосіб великий обсяг "тестованого" програмування заміняється програмуванням досить простих імітаторів модулів, що використовуються у програмі. Крім того, імітатори зручно використовувати для процесу добору тестів шляхом задання потрібних результатів, що будуть повернуті імітаторами.

Щоб проілюструвати цей принцип, розглянемо простую задачу. Обчислити і надрукувати велику степінь двійки, яка не може бути записана за допомогою типу `Integer`, наприклад 2^N.

В загальному вигляді ця програма може мати вигляд:

```
Var N: Integer;
    X: BigDig;
Begin
  Read(N);
  <Обчислити в змінній X значення 2 в степені N>
  Write('2 в степені ',N,'= ');
  <Надрукувати велике число X>
End;
```

Проте цю програму компілятор мови Pascal “зрозуміє” частково. Наступний крок в розробці програми полягає в тому, щоб “пояснити” чи деталізувати вказані інструкції. Цей процес необхідно продовжувати до тих пір поки ми не отримаємо програму в термінах мови Pascal. Таким чином будується ієрархія програм, на самому верху якої знаходиться програма записана простіше за все, а внизу знаходиться программа яку “зрозуміє” компілятор мови Pascal. Перехід з одного рівня ієрархії на більш низький можна розглядати з різних точок зору. З одного боку ми пояснюємо складну дію через більш просту, тобто розбиваємо складну задачу на декілька більш простих, які можна розв’язати відносно незалежно (в нашому прикладі задача розбита на дві окремі під задачі: обчислення великого числа та його друк). З іншого боку, кожне прийняте рішення по деталізації програми зменшує клас можливих програм для розв’язання задачі, поки цей клас

не зведеться до єдиної програми на мові паскалі. Тобто описану програму можна розглядати, як деталізацію програми самого високого рівня.

Переходячи до деталізації описаної програми, необхідно пояснити три речі: опис типу `BigDig` (велике число), інструкції обчислення 2 в степені `N` і друк великого числа. В загальному випадку спочатку бажано приймати такі рішення, які по можливості слабо залежать від ще не прийнятих рішень в інших частинах програми, окрім того, варто вибрати прості рішення з максимальними шансами на те, що їх не доведеться потім переглядати. В даному випадку почати слід з інструкції обчислення 2 в степені `N`, оскільки саме спосіб його обчислення визначає необхідні нам операції з великими числами (набір допустимих операцій і має визначати тип даних, а представлення значень вибирається з міркувань простоти і ефективності реалізації операцій). До деталізації інструкцій друку великого числа слід приступати лише після вибору представлення цього великого числа.

Обчислення 2 в степені `N` будемо проводити шляхом послідовних множень на 2. Введемо допоміжну змінну `K` і виберемо в якості інваріанту циклу умову `X=(2 в степені K)`. Фрагмент програми, що реалізує інструкцію обчислення 2 в степені `N` матиме вигляд:

```
Var K: Integer;
X:=1; { X =2 в степені 0}
For K:=1 To N Do
  Begin
    Подвоїти(X) { X=2 в степені K}
  End;
```

В цій програмі дві інструкції потребують подальшої деталізації: `Подвоїти(X)` та `X:=1`. (Остання інструкція не може бути безпосередньо виконана компілятором мови Pascal, оскільки він “не розуміє” тип змінної `X: BigDig`).

Як бачимо, подальше просування неможливе без деталізації типу великого числа. Можна бачити, що якщо реалізувати велике число просто як тип `Integer`, то інструкції подвоєння та друку зведуться до інструкцій паскаля і ми тримаємо просту програму піднесення 2 до степеня `N`, яка придатна для невеликих `N`. Проте

нас це не влаштовує. Щоб зберігати дуже великі числа, їх можна записувати в вигляді масиву, кожен елемент якого буде зберігати одну цифру числа (для запису числа 2014 буде використовуватися чотири елементи масиву, в яких розмістимо цифри 2, 0, 1, 4).

Враховуючи, що в ході роботи програмі доведеться мати справу з різними великими числами, які будуть містити різну кількість цифр, доцільно довжину масиву вибрати з розрахунку на максимальну кількість цифр. Окрім того, варто обумовити, які елементи масиву будуть зберігати цифри числа, якщо його довжина менше розміру масиву.

Згадуючи "ручний" алгоритм множення в стовпчик, можна відмітити, що цифри однакових розрядів в множників записуються одна під іншою, а добуток чисел "росте" вліво. Це наводить на думку записувати велике число в масив аналогічно, забезпечивши йому можливість "рости" вліво в сторону старших розрядів. Для цього потрібно записувати цифру одиниць в останній елемент масиву, цифру десятків – в передостанню і так далі.

Програма також має "знати", скільки цифр містить число. Цю інформацію можна отримати "пам'ятаючи" наприклад номер елементу масива, в якому починається число. Отже, представлення великого числа складається з двох частин: масив його цифр і номер елементу масиву з якого починається число. Тому нам доведеться деталізувати опис змінної великого числа `X` використовуючи дві змінні:

```
Const MaxDig=302; {число цифр достатнє для зберігання 2^1000}
Var XDig: Array[1..MaxDig] Of 0..9;
    XStart: 1..MaxDig
   { цифри числа – XDig[XStart..MaxDig] }
```

Після вибору представлення великих чисел подальше програмування розгалужується на цілком незалежні частини по реалізації окремих дій з великими числами.

Простіше за все реалізувати присвоєння змінній `X` початкового значення 1. Для цього слід надати значення 1 останньому елементу масиву та вказати в змінній `XStart`, що число `X` містить одну цифру:

```
{Оппис X:=1}
XDig[MaxDig]:=1;
XStart:=1;
```

Наступна інструкція – `Подвоїти(X)`. Будемо використовувати “ручний” алгоритм множення в стовпчик, в якому добутки отримуються по одній цифрі, починаючи з одиниць. Для вказання положення цифри, що оброблюється введемо змінну `I`, окрім того знадобиться змінна для запам’ятовування величини переносу (`CarryOver`) та допоміжна змінна `B`, в яку будемо зберігати значення подвоєної цифри. Також необхідно врахувати, що перенос зявиться тільки після обчислення добутку одиниць, тому перед виконанням обчислення потрібно змінній `CarryOver` надати значення 0. В результаті подвоєння великого числа запишеться:

```
{Оппис Подвоїти(X) }
Var I: 1.. MaxDig; {I – номер цифри що опрацбовуэться}
    CarryOver, B : Integer;
Begin
  CarryOver:=0;
  For I:= MaxDig DownTo XStart Do
   Begin
     B:=2*XDig[I]; {подвоєння цифри XDig[I]}
     B:=B+CarryOver; {додавання переносу з попереднього розряду}
     XDig[I]:=B Mod 10; {запис цифри одиниць суми в XDig[I]}
     CarryOver:= B Div 10 {запамятовування переносу}
   End;
  If CarryOver<>0 Then Begin
     XStart:= XStart-1;
     XDig[XStart]:= CarryOver; End;
End;
```

Тепер залишається реалізувати інструкцію друку великого числа `X`. Щоб надрукувати велике число, потрібно послідовно надрукувати його цифри, що зберігаються в масиві `XDig`. Реалізація даної інструкції буде мати вигляд:

```
Var I: 1.. MaxDig;
For I:= XStart To MaxDig Do
  Write(XDig[I]);
```

З’єднавши всі частини, отримаємо програму обчислення значення числа 2 в степінь N.

Зрозуміло, що кваліфікація програміста і його розуміння завдання при виконанні послідовних декомпозицій можуть значною мірою вплинути на ефективність процесу проектування. При виборі та програмуванні модулів верхніх рівнів варто враховувати не тільки функціональну значимість, але й доцільність можливості раннього отримання практично працюючих модулів. Наприклад, якщо керуюча логіка самого верхнього модуля залежить від певних змінних, то оператори, що описують та опрацьовують ці змінні, повинні бути сформовані в сегментах досить високого рівня. При цьому з’являється можливість перевірити й налагодити управління в програмі на самому початку її розробки і тим самим уникнути на цьому рівні помилок, які виправити буде найскладніше.

Наступною перевагою низхідного проектування є той факт, що в ньому практично відсутнє комплексне налагодження, що при висхідному проектуванні займає близько третьої частини від загального часу розробки програми. Перевагою розглянутого підходу є також те, що уточнені інтерфейси між модулями дають можливість отримати кістяк програми вже на перших кроках проектування; дозволяють досить легко контролювати хід роботи над проектом і варіювати ресурсами для перекриття вузьких місць.

Розробка системи може виконуватися одночасно декількома програмістами, як і при висхідному проектуванні, однак ієрархічна структура програми в значній мірі сприяє успішній взаємодії між програмістами, що співробітничають.

Проте існує і ряд недоліків низхідного проектування, які полягають у наступному [3; 22; 29].

По-перше, деяким недоліком низхідного проектування, що приводить до певних ускладнень при його застосуванні, є необхідність абстрагуватися від базових характеристик використовуваної мови програмування, видумуючи абстрактні операції, які пізніше потрібно буде реалізувати за допомогою визначених у програмі модулів, а інколи це досить складно зробити у рамках вибраної мови програмування. Однак здатність до таких абстракцій

представляється необхідною умовою розробки великих ПЗ, тому її потрібно розвивати.

По-друге, строге слідування принципам цього підходу може привести до того, що процес розробки всієї програми просунеться досить далеко, перш ніж з'ясується, що компоненти нижніх рівнів не можуть виконати покладені на них функції (наприклад, обробити вхідні сигнали в реальному масштабі часу або забезпечити потрібну точність). У цьому випадку необхідно буде повернутися до верхніх рівнів і повторити проектування і структурування.

По-третє, оскільки модулі можуть проектуватися незалежно різними програмістами, можливі ситуації, коли всередині двох або більшого числа різних модулів виникнуть потреби в однакових обчисленнях які могли б реалізуватися однієї й тією ж підпрограмою. Однак ця можливість може бути не використана через те, що програмісти можуть цього не знати або довідаються занадто пізно, так що можуть знадобитися дещо більші переробки. Цей недолік у значній мірі переборюється при правильному застосуванні наскрізного структурного контролю.

У класичному низхідному підході рекомендуються спочатку запрограмувати всі модулі, а вже потім починати їх низхідне тестування. Однак такий порядок розробки не вбачається досить обґрунтованим: тестування й налагодження модулів може привести до зміни специфікації підлеглих модулів і навіть до зміни самої модульної структури програми, так що в цьому випадку програмування деяких модулів може виявитися даремно проробленою роботою.

Вбачається більш раціональним інший порядок розробки програми, відомий у літературі як ***підхід спадної реалізації*** [22]. У цьому підході тестування кожного модуля починається відразу, після того, як він був запрограмований, і лише після тестування даного модуля відбувається перехід до програмування іншого модуля.

У розглянутих підходах висхідного і низхідного проектування та розробок (які називаються класичними) деревоподібна структура програми повинна розроблятися до початку програмування модулів. Однак в такому випадку сумнівним є те, що до програмування модулів можна розробити структуру

програми досить змістовно й точно. Насправді такий підхід, розробки структури програми до початку програмування модулів, не є обов'язковим. Так існують, підходи, в яких модульна структура формується в процесі програмування модулів, такими підходами є конструктивний та архітектурний.

Конструктивний підхід проектування та розробки представляє собою модифікацію спадної розробки, при якій деревоподібна модульна структура програми формується в процесі програмування окремих модулів. спочатку програмується головний модуль, виходячи зі специфікації програми в цілому, причому специфікація програми є одночасно й специфікацією її головного модуля, тому що на нього покладено відповідальність за виконання функцій програми. У процесі програмування головного модуля, якщо він виявляється досить великим, виокремлюються підзадачі (внутрішні функції), у термінах головного модуля. Це означає, що для кожної виокремленої підзадачі створюється специфікація її фрагмента програми, що надалі може бути представлений деяким піддеревом модулів. Важливо відмітити, що відповідальність за виконання визначених функції покладено на головний (може бути, і єдиний) модуль цього піддерева, так що специфікація визначеної функції є одночасно й специфікацією головного модуля цього піддерева. У головному модулі програми для звернення до визначеної функції формується звернення до головного модуля зазначеного піддерева і відповідно до створеної його специфікації. Таким чином, на першому кроці розробки програми (при програмуванні її головного модуля) формується верхня початкова частина дерева.

Аналогічні дії проводиться при програмуванні інших модулів, що вибираються з поточного стану дерева програми із числа специфікованих, але поки ще не запрограмованих модулів. У результаті цього відбувається чергове перетворення дерева програми.

Архітектурний підхід проектування та розробки представляє собою модифікацію висхідної розробки, у якій деревоподібна модульна структура програми формується в процесі програмування окремих модулів. Це означає, що для заданої предметної галузі визначаються типові функції, кожна з яких може

використовуватися при розв’язанні різних завдань в межах цієї галузі, а потім специфікуються та програмуються окремі програмні модулі, що виконують ці функції. Оскільки процес визначення таких функцій пов’язаний з нагромадженням і узагальненням досвіду розв’язання завдань у заданій предметній галузі, то зазвичай спочатку визначаються й реалізуються окремими модулями більш прості функції, а потім поступово програмуються модулі, що використовують раніше визначені функції. Такий набір модулів створюється в розрахунку на те, що при розробці тієї або іншої програми в межах деякої предметної галузі можуть виявитися прийнятними деякі із цих модулів. Це надає можливість істотно скоротити затрати праці на розробку конкретної програми шляхом підключення до неї заздалегідь заготовлених і перевірених на практиці модулів. Оскільки такі модулі можуть багаторазово використовуватися в різних конкретних програмах, то архітектурний підхід може розглядатися як шлях боротьби з дублюванням коду у програмуванні. У зв’язку із цим програмні модулі, що створені в рамках архітектурного підходу, зазвичай параметризуються для того, щоб підсилити застосовність таких модулів шляхом зміни параметрів.

Всі ці підходи мають і інші різновиди залежно від того, у якій послідовності обходяться модулі деревоподібної структури програми в процесі її розробки. Це можна робити, наприклад, по рівнях, розробляючи всі модулі одного рівня, перш ніж переходити до наступного рівня. При спадній розробці дерево можна обходити також у лексикографічному порядку (зверху-вниз, зліва-на право). При конструктивній реалізації для обходу дерева програми доцільно додержуватися ідей Фуксмана, які він використовував у запропонованому їм підході вертикального шарування [236, с. 88]. суть такого обходу полягає в тому, що в рамках конструктивного підходу спочатку реалізуються тільки ті модулі, які необхідні для самого найпростішого варіанта програми, що може нормально використовуватися для досить обмеженої кількості наборів вхідних даних, але для таких даних розв’язання буде відбуватися до кінця. Замість інших модулів, на які в даній програмі є посилання, вставляються лише їх імітатори, що забезпечують, в основному, контроль за виходом за межі даного завдання. Потім до цієї програми

додаються реалізації деяких інших модулів (зокрема, замість деяких з наявних імітаторів), що забезпечують потрібне опрацювання деяких інших наборів вхідних даних. і цей процес триває поетапно до повної реалізації необхідної програми. Такий обхід дерева програми відбувається з метою найкоротшим шляхом реалізувати той або інший варіант нормально діючої програми, проте

Підбиваючи підсумок сказаному, на рисунку 1.1 наведено загальну схему класифікації розглянутих підходів проектування та розробки ПЗ.

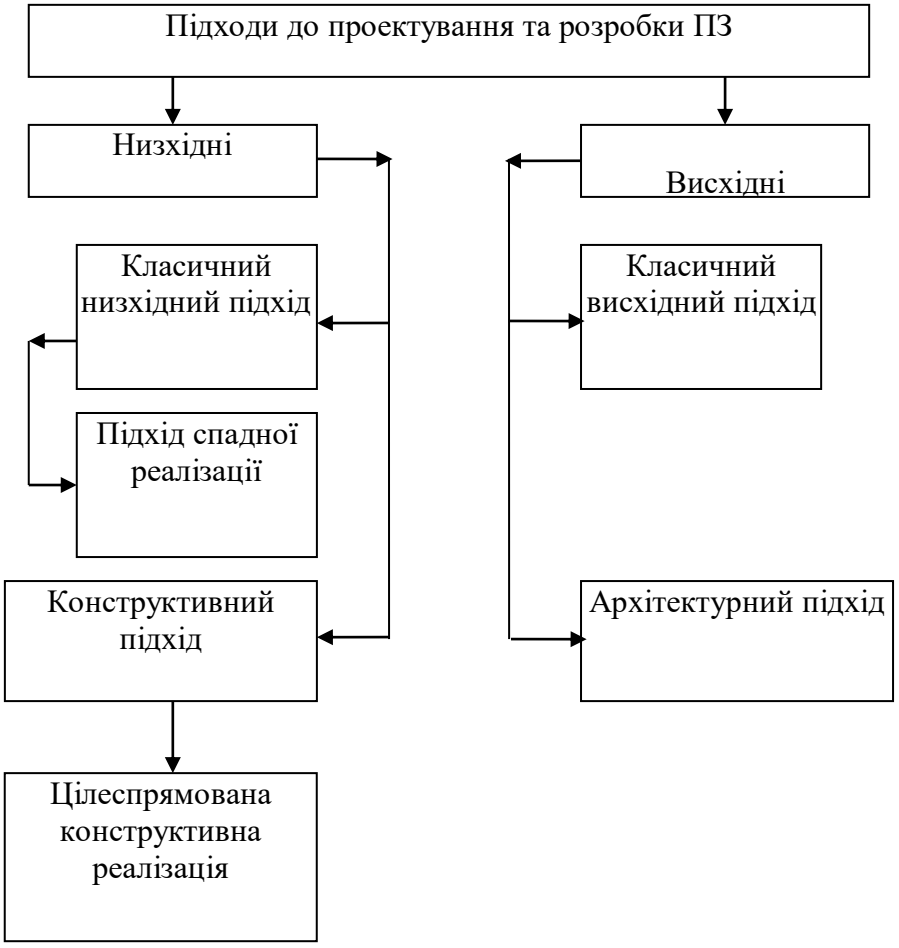


Рис. 1.1. Класифікація підходів розробки структури ПЗ

Зрозуміло, що програміст при створенні ПЗ на різних етапах може використовувати різні підходи, використовуючи один підхід при визначенні підмодулів скористатися зовсім іншим підходом на іншому етапі в залежності від поточних потреб.

1.3.3. Технології програмування. Крім парадигми програмування важливою для програміста є оволодіння технологіями програмування.

Під технологією програмування за Бежановим М.М. під технологією програмування розуміють сукупність узагальнених і систематизованих знань, або науку, про виважені способи (прийоми і процедури) проведення процесу програмування, що забезпечує в заданих умовах отримання програмної продукції із заданими властивостями [15].

Технологія програмування визначає деяку професійну культуру роботи фахівців (не тільки програмістів), що забезпечує заданий рівень продуктивності праці і якості отримуваної в результаті програмної продукції.

До сучасних технологій програмування ПЗ можна встановити такі вимоги [100, с. 56]:

- технологія програмування не повинна бути пов’язана з мовою програмування;
- технологія програмування повинна забезпечувати відмежованість ПЗ від конкретного розробника, тобто людський фактор у програмуванні повинен бути зведений до мінімуму. Це необхідно як при розробці ПЗ, так і при його супроводі та модифікації для інших умов експлуатації;
- засоби автоматизації технології програмування повинні охоплювати всі етапи роботи розробника. Вони повинні враховувати існуючий досвід, відображений у вітчизняних та зарубіжних стандартах, а також повинні забезпечувати можливість гнучкого й простого переналагодження на основі досвіду розробника, що постійно накопичується;
- технологія програмування повинна забезпечувати цілеспрямовану роботу колективу програмістів, а не окремих особистостей;
- технологія програмування повинна бути простою в освоєнні.

Серед технологій програмування можна виокремити структурну, модульну та об’єктно-орієнтовну.

В основі с***труктурної технології програмування*** лежить представлення програми в вигляді ієрархічної структури блоків. структурне програмування є втіленням принципів системного підходу в процесі створення та експлуатації ПЗ. В основу структурного програмування покладені такі положення:

- будь-яка програма представляє собою структуру, побудовану з трьох типів базових конструкцій (послідовне виконання, розгалуження, цикл);
- фрагменти програми, що повторюються, можуть оформлюватися в вигляді так званих підпрограм;
- розробка програми відбувається покроково.

Фундаментом структурного програмування є доведена Коррадо і Якопіні теорема про структурування [247]. Ця теорема встановлює, що яким би складним не було завдання, блок-схема відповідної програми завжди може бути представлена з використанням досить обмеженого числа елементарних керуючих структур (розгалуження, повторення, та ін.). Ці елементарні структури можуть з'єднуватися між собою, створюючи більш складні структури, за тими ж самими елементарними схемами.

На початковому етапі структурного програмування здійснюється так званий структурний аналіз, який полягає у виокремленні окремих структур програми, що надає можливість всебічно проаналізувати окремі елементи створюваної програми та їх взаємозв'язок. структурним аналізом також прийнято називати метод дослідження системи, що починається з її загального огляду, а потім деталізації, отримуючи ієрархічну структуру із все більшим числом рівнів. Для таких методів характерна розбивка на рівні абстракції з обмеженим числом елементів на кожному рівні; обмеженість контексту, що включає лише істотні на кожному рівні деталі; дуальність даних і операцій над ними; використання строгих формальних правил запису; послідовність наближення до кінцевого результату.

Найпоширеніші методології структурного підходу базуються на низці загальних принципів, частина з яких регламентує організацію робіт на початкових етапах розробки ПЗ, а частина використовується при розробці рекомендацій з організації робіт. У якості базових принципів можна визначити наступні [22; 100]:

- принцип "розділяй і пануй" – базується на розбитті складних завдань на менші незалежні завдання, що є більш легкими для розуміння і розв'язання;

- принцип ієрархічного упорядкування – базується на організації складових частин майбутньої програми в ієрархічні деревоподібні структури з додаванням нових деталей на кожному рівні;
- принцип абстрагування – полягає у відокремленні істотних аспектів системи й нехтування несуттєвими;
- принцип формалізації – полягає в необхідності строгого підходу до розв'язання проблеми;
- принцип несуперечності – полягає в обґрунтуванні й узгодженні елементів і перевірки їх на несуперечність;
- принцип незалежності даних – полягає в тому, що моделі даних повинні бути проаналізовані й спроектовані незалежно від процесів їхнього логічного опрацювання, а також від їхньої фізичної структури і розподілу;
- принцип приховування – полягає в приховуванні несуттєвих на конкретному етапі даних;
- принцип логічної незалежності – полягає в концентрації уваги на логічному проектуванні для забезпечення незалежності від фізичного проектування.

Дотримання зазначених принципів необхідне при організації робіт на початкових етапах розробки ПЗ незалежно від типу ПЗ, що розроблюється, й використовуваних при цьому методологій. Керуючись всіма принципами в комплексі, можна на більш ранніх стадіях розробки зрозуміти, що представлятиме собою створюваний ПЗ, а також виявити помилки й недоліки на ранніх етапах проектування, що, у свою чергу, полегшить роботи на наступних етапах розробки ПЗ і знизить витрати часу.

Модульна технологія програмування полягає в розбитті програми на відносно незалежні логічні частини, які називають програмними модулями. Програмний модуль – це будь-який фрагмент опису виконуваних операторів, оформлений як самостійний програмний продукт, який можна використовувати в описах інших модулів [89]. Тобто кожний програмний модуль програмується, компілюється й налагоджується окремо від інших модулів програми. Будемо

розглядати програмний модуль в широкому розумінні тобто, як сукупність підпрограм. Наприклад в мовах програмування Pascal програмними модулями є файли з розширенням tpu. Проте інколи в літературі зустрічається і поняття програмного модуля, в вузькому розумінні, як єдина окрема підпрограма.

Кожний розроблений програмний модуль може включатися до складу різних програм, за умови виконання умов його використання. Таким чином, програмний модуль, а отже і модульна технологія програмування, може розглядатися як засіб боротьби зі складністю програм, і як засіб боротьби з дублюванням у програмуванні.

Програмний модуль характеризують: функціональною завершеністю – модуль виконує перелік регламентованих операцій для реалізації кожної окремої функції в повному обсязі; логічною незалежністю – результат роботи програмного модуля залежить тільки від вхідних даних, і не залежить від роботи інших модулів; слабкістю інформаційних зв’язки з іншими програмними модулями – обмін даними між модулями повинен бути мінімізований, наскільки це можливо; осяжністю щодо розміру й складності.

Таким чином, програмний модуль має містити визначення доступних для опрацювання даних, операції опрацювання даних, схеми взаємозв’язку з іншими модулями. склад і вид програмних модулів, їхнє призначення й характер використання в програмі в значній мірі визначаються інструментальними засобами.

Проте не всякий програмний модуль сприяє спрощенню програми. Виокремити прийнятний із цього погляду модуль є серйозним завданням. Для оцінки прийнятності виокремленного модуля можуть бути використані деякі критерії.

Так, Хольт [249] запропонував такі два загальних критерії:

- хороший модуль зовні простіше, ніж всередині;
- хороший модуль простіше використовувати, ніж побудувати.

Майерс пропонує використовувати більш конструктивні характеристики програмного модуля для оцінки його прийнятності, а саме: розмір, міцність модуля та зчеплення модуля [153, с. 92-113].

Розмір модуля вимірюється числом операторів, що містяться в ньому. Модуль не повинен бути занадто малим або занадто великим. Занадто малі модулі приводять до громіздкої модульної структури програми. Занадто великі модулі незручні для розуміння та внесення змін, вони можуть істотно збільшити сумарний час при налагодженні програми. Зазвичай рекомендуються програмні модулі розміром від декількох десятків до декількох сотень операторів.

Міцність модуля – це міра його внутрішніх смислових та логічних зв'язків, тобто розміщення в модулі підпрограм, що логічно зв'язані по функціональному використанню, відносяться розробником до одного класу задач чи спільністю вхідних даних. Наприклад, сукупність підпрограм, що реалізують чисельні методи знаходження наближеного значення кореня функції.

Міцність модуля зменшується, коли в модулі об'єднуються підпрограми, що можуть використовуватися для досить різних, незв'язаних, задач. Не міцні модулі можуть з'являтися в результаті переведення існуючого монолітного коду в модулі.

Зчеплення модуля – це міра його залежності за даними від інших модулів. Тобто зчеплення модулів зростає, виникає, коли один модуль використовує дані іншого модуля і може змінювати ці дані, коли два чи більше модулів посилаються на одну й ту саму глобальну структуру даних чи глобальні змінні.

Чим слабкіше зчеплення модуля з іншими модулями, тим сильніше його незалежність від інших модулів. У свою чергу чим більша незалежність модуля від інших модулів, тим легше розібратися в його коді, легше організувати розробку ПЗ колективом програмістів, менша ймовірність появи нових помилок при використанні раніше створених модулів. Для досягнення більшої незалежності модулів передавання даних між модулями необхідно проводити в формі явних і простих параметрів.

Ступінь міцності і зчеплення можна використовувати для оцінювання існуючого проекту і як керівний принцип при проектуванні нової програми. Висока міцність і слабке зчеплення сприяють незалежності модулів, оскільки вони зводять до мінімуму їхню взаємодію і взаємозалежність.

Об'єктно-орієнтовна технологія програмування полягає в представленні програми у вигляді сукупності об'єктів, кожен з яких є екземпляром певного класу (класи утворюють ієрархію наслідуванням) і має інтерфейс у вигляді набору методів для взаємодії один з одним.

З переходом до об'єктно-орієнтованої технології, різкі границі між аналізом і проектуванням починають зникати, поступово стаючи невід'ємною властивістю методології. Але, незважаючи на це, аналіз і проектування важливо розглядати як різні процеси. Зрозуміло, вони можуть частково перетинатися й проводитися одночасно, але в той же час, це – різні етапи.

На початковому етапі застосування об'єктно-орієнтованої технології програмування відбувається, так званий об'єктно-орієнтований аналіз, який починається з дослідження предметів чи явищ реального світу, що є частиною задачі, що розв'язується. Ці предмети чи явища, які можуть називатися об'єктами, індивідуально характеризуються атрибутами стану (даних, що зберігається в змінних) і операторами опрацювання даних (поведінкою). Використовуючи об'єктно-орієнтовану термінологію, відбувається формулювання й опис класів, що охоплюють предметну галузь. Одночасно з описом цих індивідуальних характеристик також моделюються зв'язки або взаємодії між об'єктами предметної галузі. Ці зв'язки можуть встановлюватися у формі агрегування частин (це є частиною того), делегування (це використовує те) або наслідування (це є нащадком того).

У подальшому після застосування об'єктно-орієнтованого аналізу відбувається перехід від моделювання предметної галузі до моделювання області реалізації. структура класу тепер починає включати опис специфічних комп'ютерних об'єктів. Наприклад: класи інтерфейсу користувача (вікна, меню, і т.д.), класи керування завданнями (процеси, і т.д.), класи даних (списки, стеки,

черги, і т.д.). Оскільки об’єктно-орієнтований аналіз і проектування використовують ту ж саму мову і можуть використовувати ті ж самі системи позначень, простіше і доцільніше виконувати обидва процеси паралельно й ітераційно. Як вказує Гради Буч (Grady Booch): «Границі між аналізом і проектуванням розмиті, хоча фокус кожного повністю різний. При аналізі, ми намагаємося моделювати світ, виявляючи класи й предмети, які формують словник предметної галузі, а при проектуванні, ми виокремлюємо абстракції й механізми, які забезпечують поводження, які потребує ця модель» [28].

Надзвичайно важливо відзначити, що цілі процесу аналізу й процесу проектування не зосереджені винятково на процесі пошуку розв’язків, необхідних для поточного розуміння проблеми. скоріше, вони спрямовані на проектування й формування узагальнених класів із закінченими і корисними структурами.

Об’єктно-орієнтований аналіз, проектування і програмування при спільній роботі підсилюють загальний ефект, забезпечуючи отримання розв’язків, які краще моделюють предметну галузь, ніж подібні системи, що створені на основі структурного підходу. Змодельовані системи простіше адаптувати до нових умов, легше модернізувати, вони є більш стійкими і підтримують багаторазове використання коду. Причини цих покращень полягають у наступному: об’єктна орієнтація базується на більш високому рівні абстракції; об’єктно-орієнтоване проектування й програмування заохочують багаторазове використання коду.

Граді Буч сформулював головну перевагу об’єктно-орієнтованого підходу у такий спосіб: “об’єктно-орієнтовані системи більш відкриті та легко піддаються внесенню змін, оскільки їх конструкція базується на стійких формах. Це дає можливість системі розвиватися поступово й не приводить до повної її переробки навіть у випадку істотних змін вихідних вимог” [28, с. 13].

Буч відзначив також ряд наступних переваг об’єктно-орієнтованого підходу [28]:

- можливість створювати програмні системи меншого розміру шляхом використання загальних механізмів об’єктно-орієнтованого підходу;

- підвищення рівня уніфікації розробки при повторному використанні вже розроблених систем. системи найчастіше виходять більше компактними, ніж їх не об’єктно-орієнтовані еквіваленти, що означає не тільки зменшення обсягу програмного коду, але й здешевлення проекту за рахунок використання попередніх розробок;
- системи є більш гнучкими і простіше еволюціонують з часом, тому що їх схеми базуються на стійких проміжних формах. Тобто відбувається зниження ризику при створенні складних програмних систем, оскільки вони розвиваються на менших системах, в яких розробник вже впевнений;
- об’єктна модель є цілком природною, оскільки в першу чергу орієнтована на людське сприйняття світу, а не на комп’ютерну реалізацію;
- використання об’єктної моделі надає можливість повною мірою використовувати особливості об’єктно-орієнтованих мов програмування.

До недоліків об’єктно-орієнтованого підходу відноситься деяке зниження продуктивності функціонування ПЗ і високі початкові часові затрати.

На нашу думку на початкових етапах навчання програмування варто застосовувати процедурні мови програмування, які є більш зрозумілими у використанні. Зокрема враховуючи строгість мови Pascal, її можна вважати доцільною при навчанні програмування в педагогічних ВНЗ. У подальшому вивчення програмування і зокрема навчання створення ППЗ варто базувати на парадигмі об’єктно-орієнтованого програмування і використанні відповідних об’єктно-орієнтованих мов, зокрема Object Pascal та середовищі програмування Lazarus.

Що стосується технологій програмування, то можна зазначити, що розглянуті технології мають між собою спільні риси. Так основою взаємозв’язку між структурною і об’єктно-орієнтованою технологією є спільність ряду категорій і понять: процес і варіант використання, сутність і клас та ін. Цей взаємозв’язок може проявлятися в різних формах. Так, одним з можливих варіантів є використання структурного аналізу як основи для об’єктно-

орієнтованого проектування. Після виконання структурного аналізу можна різними способами перейти до визначення класів і об’єктів. У свою чергу використання програмних модулів досить суттєво використовується в сучасних об’єктно-орієнтованих мовах програмування. Отже в процесі підготовки до розроблення ППЗ за основу варто брати об’єктно-орієнтовану технологію програмування, яка буде включати в себе окремі елементи як структурної так і модульної технології програмування.

Окрім того, варто відмітити, що підготовка майбутніх учителів до розроблення ППЗ має базуватися на використанні саме об’єктно-орієнтованої парадигми програмування, оскільки безперечною її перевагою є концептуальна близькість до предметної галузі довільної структури та призначення. Використання механізму наслідування атрибутів і методів надає можливість будувати похідні системи на основі базових, і таким чином, створювати моделі предметної галузі будь-якої складності з заданими властивостями. Ще однією теоретично цікавою і практично важливою властивістю об’єктно-орієнтованого підходу є підтримка механізму опрацювання подій, які змінюють атрибути об’єктів і моделюють їх взаємодію в предметній галузі. Переміщаючись по ієрархії класів від загальних понять предметної галузі до більш конкретних (або від більш складних – до більш простих) і навпаки, програміст отримує можливість змінювати ступінь абстрактності або конкретності погляду на модельований ним реальний світ. Використання раніше розроблених (можливо, іншими колективами програмістів) бібліотек класів і методів надає можливість значно заощадити трудовитрати при розробці ПЗ. Об’єкти, класи і методи можуть бути поліморфними, що робить реалізоване програмне забезпечення більш гнучким і універсальним. Проте складність несуперечливої і повної формалізації об’єктної теорії породжує труднощі тестування та верифікації створеного програмного забезпечення. Мабуть, ця обставина є одним з найбільш істотних недоліків об’єктно-орієнтованого підходу до програмування.

1.4. Особливості проектування педагогічних програмних засобів

1.4.1. Вимоги до ППЗ. Визначення вимог до ПЗ виражає в абстрактній формі потреби користувача і є початковим завданням для розробки ПЗ. Вимоги загалом визначають задум ПЗ, характеризують умови його використання. Тому розробка ПЗ починається з встановлення вимог, що досить повно характеризують потреби користувача й надає можливість розробнику адекватно сприймати ці потреби.

При складанні вимог вони можуть бути представлені у вигляді певних текстових фрагментів, написаних природною мовою, таблиць та діаграм з додаванням описів щодо особливостей використання окремих елементів програми. Формульовані вимоги повинні бути зрозумілими тому кому вони призначаються.

Відомі три способи визначення вимог до ПЗ [153]:

- вимоги керовані користувачем;
- вимоги контрольовані користувачем;
- вимоги незалежні від користувача.

Вимоги керовані користувачем, визначаються користувачем чи групою користувачів майбутнього ПЗ. Роль розробника ПЗ у визначенні цих вимог зводиться, в основному, до з’ясування того, наскільки зрозумілі йому ці вимоги. Такими вимогами можуть бути: вимоги щодо необхідних характеристик чи елементів інтерфейсу ПЗ.

Вимоги контрольовані користувачем, формулюються розробником при участі користувача чи групи користувачів. Роль користувача в цьому випадку зводиться до інформування розробника про свої потреби в ПЗ і контролю за тим, щоб визначені вимоги дійсно виражали його потреби. Такими вимогами можуть бути вимоги які стосуються естетичного та ергономічного вигляду ПЗ.

Вимоги незалежні від користувача, визначаються без якої-небудь участі користувача (на повну відповідальність розробника). Такими вимогами можуть бути вимоги, які стосуються технічної сторони ПЗ.

Загальновідомо, що розробка програм, які використовуються в навчальних цілях, являє собою дуже складний процес, що вимагає колективної праці не тільки вчителів, методистів, програмістів, але й психологів, гігієністів, дизайнерів.

Згідно тимчасових вимог до ППЗ для загальноосвітніх, професійно-технічних і вищих навчальних закладів колектив розробників повинен включати фахівців таких груп [183]:

- наукова, педагогічна та методична – наукові працівники, вчителі-методисти, консультанти закладів освіти;
- програмно-апаратна та організаційна – фахівці зі створення та впровадження інформаційних і комп'ютерних технологій, фахівці з мережевого, системного, програмного, апаратного, прикладного забезпечення, фахівці з комп'ютерної графіки та дизайну, художники;
- нормативно-виробнича – фахівці з метрології, стандартизації та сертифікації, психологи, спеціалісти з ергономіки.

У зв'язку із цим правомірно пред'являти комплекс вимог до створюваних ППЗ, щоб їх використання не викликало б негативних (у психолого-педагогічному або фізіолого-гігієнічному сенсі) наслідків, а служило б цілям інтенсифікації навчального процесу, розвитку особистості учня.

Виходячи з цього, в [227] перераховано основні вимоги, що висуваються до ППЗ: дидактичні вимоги, методичні вимоги, обґрунтування вибору тематики, психологічні, ергономічні вимоги, естетичні вимоги, програмно-технічні вимоги, вимоги до оформлення документації.

Дидактичні та методичні вимоги можуть розглядатися, як єдине ціле – педагогічні вимоги, які повинні мати найбільший вплив при створенні ППЗ, тому їх зміст розкриємо більш детально.

Дидактичні вимоги до ППЗ виражаються в наступних принципах: науковості, доступності, адаптивності, систематичності та послідовності, комп'ютерної візуалізації навчальних даних, самостійності, активізації навчальної діяльності, міцності засвоєння результатів навчання, зворотного зв'язку, розвитку інтелектуального потенціалу.

Вимоги забезпечення *науковості* змісту ППЗ передбачають представлення засобами програми науково-достовірних відомостей. При цьому можливість моделювання, імітації досліджуваних об’єктів, явищ, процесів (як реальних, так і "віртуальних") може забезпечити проведення експериментально-дослідницької діяльності, що ініціює самостійне "відкриття" закономірностей досліджуваних процесів, і разом з тим наближає шкільний експеримент до сучасних наукових методів дослідження.

Вимога забезпечення *доступності* означає, що навчальний матеріал, запропонований ППЗ, форми й методи організації навчальної діяльності повинні відповідати рівню підготовки учнів і їх віковим особливостям. Встановлення того, чи доступний розумінню учня запропонований за допомогою ППЗ навчальний матеріал, чи відповідає він раніше набутим знанням, умінням і навичкам, визначається вчителем. Від отриманих результатів залежить подальший хід навчання з використанням ППЗ.

Вимога *адаптивності* (пристосованості ППЗ до індивідуальних можливостей учня) передбачає реалізацію індивідуального підходу до учня, врахування індивідуальних можливостей сприйняття навчального матеріалу. Реалізація адаптивності може забезпечуватися різними засобами наочності, декількома рівнями диференціації при представленні навчального матеріалу за складністю, обсягом чи змістом.

Вимога забезпечення *систематичності та послідовності* навчання з використанням ППЗ передбачає необхідність засвоєння учнем системи понять, фактів і способів діяльності в їх логічному зв’язку з метою забезпечення послідовності й наступності в оволодінні компетентностями.

Вимога забезпечення *комп’ютерної візуалізації навчальних даних*, передбачає реалізацію можливостей сучасних засобів візуалізації (наприклад, засобів комп’ютерної графіки, технології мультимедіа), об’єктів, процесів, явищ (як реальних, так і "віртуальних"), а також їх моделей, подання їх динаміки розвитку в часовому та просторовому русі, зі збереженням можливості користувача

управляти програмою шляхом уточнень певних параметрів за рахунок допоміжних вікон.

Вимога забезпечення *самостійності* передбачає забезпечення засобами програми можливих самостійних дій учня щодо здобуття навчальних даних при чіткому розумінні конкретних цілей і завдань навчальної діяльності.

Вимога забезпечення *активізації навчальної діяльності* передбачає забезпечення можливості самостійного керування ситуацією на екрані, вибором режиму навчальної діяльності; варіативністю дій у випадку ухвалення самостійних рішень; створенням позитивних стимулів, що спонукають до навчальної діяльності та підвищують мотивацію навчання (наприклад, часткове застосування ігрових ситуацій, гумор, доброзичливість інтерфейсу).

Вимога забезпечення *міцності засвоєння результатів навчання* передбачає забезпечення усвідомленого засвоєння учнем змісту, внутрішньої логіки і структури навчального матеріалу, що надаються засобами ППЗ. Ця вимога досягається здійсненням самоконтролю й самокорекції; забезпеченням контролю на основі зворотного зв’язку, з діагностикою помилок за результатами навчання і оцінкою результатів навчальної діяльності, поясненням сутності допущеної помилки; тестуванням, що констатує просування в навчанні.

Вимога забезпечення *зворотного зв’язку* передбачає необхідність організації зворотного зв’язку ППЗ, забезпечення можливості вибору варіантів змісту досліджуваного матеріалу, а також реакції програми на дії користувача, одержання допомоги, рекомендації або пояснення висунутої гіпотези.

Вимога *розвитку* і*нтелектуального потенціалу* учня передбачає забезпечення розвитку мислення (наприклад, алгоритмічного, програмістського стилю, наочно-образного, теоретичного); формування вміння приймати виважені рішення або варіативні рішення в складній ситуації; формування вмінь з опрацювання даних (наприклад, на основі використання систем опрацювання даних, інформаційно-пошукових систем, баз даних).

Методичні вимоги до ППЗ передбачають необхідність: враховувати своєрідність і особливості конкретного навчального предмета, специфіку

відповідної науки, її понятійного апарату, особливості методів дослідження її закономірностей; реалізації сучасних методів опрацювання даних.

Обґрунтування вибору тематики ППЗ. При розробці ППЗ необхідно аргументувати його педагогічну доцільність використання та відповідність методичним вимогам досягнення яких можливе лише при використанні ППЗ.

Психологічні вимоги до ППЗ передбачають необхідність: врахування вікових і індивідуальних особливостей учнів, різних типів організації нервової діяльності, різних типів мислення, закономірностей відновлення інтелектуальної й емоційної працездатності; забезпечення підвищення рівня мотивації навчання і позитивних стимулів.

Ергономічні вимоги до змісту й оформлення ППЗ обумовлюють необхідність: забезпечувати підвищення рівня мотивації навчання, позитивні стимули при взаємодії учня з ППЗ (доброзичлива й тактовна форма звертання до учня, можливість кількаразового звертання до програми у випадку невдалої спроби, можливість використання в програмі ігрових ситуацій); встановлювати вимоги до відображення даних (кольорова гама, розбірливість, чіткість зображення), до ефективності сприйняття зображення, до розташування тексту на екрані ("віконне", табличне, у вигляді тексту, що заповнює весь екран, і т.д.), до режимів роботи з ППЗ.

Естетичні вимоги до ППЗ встановлюють: відповідність естетичного оформлення функціональному призначенню; відповідність кольорового колориту призначенню ППЗ і ергономічним вимогам; упорядкованість і виразність графічних і образотворчих елементів ППЗ.

Окремі елементи ергономічних та естетичних вимог обумовлюють створення зручного та зрозумілого інтерфейсу користувача.

Програмно-технічні вимоги до ППЗ передбачають вимоги щодо забезпечення: стійкості до помилкових, некоректних та несанкціонованих дій користувача; мінімізації часу на дії користувача; ефективного використання технічних ресурсів (у тому числі і зовнішній пам’яті); відповідності функціонування ППЗ опису в експлуатаційній документації.

Вимоги до оформлення документації на розробку й використання ППЗ встановлюють єдиний порядок побудови і оформлення основних документів на розробку й використання ППЗ, які створюються в установах і організаціях незалежно від їхньої відомчої приналежності.

Крім навелених вимог до ППЗ можна сформулювати і деякі принципи щодо організації ППЗ [183]:

- *відкритість,* інтерфейс ППЗ має бути відкритим для взаємодії з іншими інформаційними системами;
- *сумісність,* шляхом узгодження змістової частини ППЗ з міжнародними, державними і галузевими (освітянськими) стандартами;
- *орієнтація інструментальних засобів на користувача:* ППЗ повинен бути простим у використанні і доступним для оволодіння людиною, яка має лише загальні навички роботи з комп'ютером;
- *об'єктна організація вмісту,* змістова частина ППЗ повинна представлятися у вигляді окремих об'єктів, що дозволить структурувати дані, забезпечити каталогізацію і пошук об'єктів за їх властивостями, багаторазово використовувати раніше створені об'єкти;
- забезпечення прав інтелектуальної власності розробника та замовника ППЗ.

1.4.2. Показники якості ППЗ. Розробка програмного забезпечення − це, насамперед, знаходження способів отримання якісного програмного продукту. Якість ПЗ може вимірятися в зовнішніх характеристиках (наприклад, легкий у використанні, виконується швидко) або у внутрішніх характеристиках (наприклад, модульна конструкція, зручний для читання код). Зовнішні метрики для більшості користувачів мають більше значення. Користувачу насправді мало цікаво, наскільки хорошу модульну конструкцію було використано при створенні програми, більшість тільки турбує, щоб ПЗ добре працював. Однак, внутрішні (сховані) метрики є ключем до створення зовнішніх, і при цьому необхідно враховувати правила конструювання програм і техніку програмування.

Процес забезпечення якості ПЗ є процесом забезпечення відповідних гарантій того, що ПЗ відповідає встановленим вимогам. Саме поняття якості ПЗ визначається як сукупність властивостей ПЗ, які обумовлюють його придатність задовольняти заданим потребам в відповідності з його призначенням [239]. Якість ПЗ оцінюється великою кількістю різноманітних характеристик, які в свою чергу, включать в себе ще цілий ряд підхарактеристик.

Розробка специфікації якості зводиться до побудови своєрідної моделі якості створюваного ПЗ [153; 89]. У цій моделі повинен бути перелік всіх тих властивостей, які потрібно забезпечити в створюваному ПЗ. При цьому кожна із цих властивостей повинна бути в достатній степені конкретизована для можливості її оцінки в розробленому ПЗ.

Добір показників якості ПЗ є достатньо складним процесом. існує досить багато класифікацій показників якості ПЗ. Так в міжнародному стандарті ISO 9126:2001 [259] визначений універсальний для будь якого класу ПЗ набір із шести показників якості:

- функціональність (functionality) – надання користувачу можливості, при заданих умовах, вирішувати потрібні йому задачі в межах відповідної предметної галузі;
- надійність (reliability) – надання користувачу можливості безвідмовного використання при заданих умовах за встановлений період часу;
- зручність використання (usability) або практичність – надання користувачу можливості зручного використання;
- продуктивність (efficiency) або ефективність – можливість, при заданих умовах, забезпечувати необхідну працездатність стосовно визначених для нього ресурсів;
- зручність супроводу (maintainability) – зручність проведення всіх видів діяльності, пов’язаних із супроводом програми;
- мобільність (portability) – збереження працездатності при перенесенні з одного оточення в інше, враховуючи організаційні, апаратні й програмні

аспекти оточення. іноді ця характеристика називається у нашій літературі мобільністю.

Згідно іншого міжнародного стандарту ISO 9126:2004 [258] пропонується більш вузький набір характеристик:

- ефективність (effectiveness) – надання користувачу можливості вирішувати його задачі конкретної предметної галузі з необхідною точністю;
- продуктивність (productivity) – надання користувачу необхідних результатів в рамках очікуваних витрат ресурсів;
- безпека (safety) – забезпечення низького рівня ризику завдання втрат життю й здоров'ю людей, бізнесу, власності або навколишньому середовищу;
- задоволення користувачів (satisfaction) – можливість приносити задоволення (комфорт, прийнятність використання, корисність) користувачам при використанні ПЗ в заданій предметній галузі.

Взявши за основу стандарт якості ПЗ ISO 9126:2001, сформулюємо перелік показників якості ППЗ та розкриємо їх зміст. Розглянуті показники можуть бути поділені на дві групи: показники, які в більшій мірі стосуються користувачів – надійність, функціональність, зручність використання або практичність, ефективність або продуктивність; та якості, які в більшій мірі стосуються програмістів – мобільність, легкість у освоєнні тексту програми, модифікованість.

Надійність – надання користувачу можливості безвідмовно виконувати визначені функції при заданих умовах протягом заданого періоду часу за допомогою ППЗ. При цьому під відмовою виконання розуміють виникнення в ППЗ помилок. Таким чином, надійний ППЗ не виключає наявність в ньому помилок – важливо лише, щоб ці помилки при його практичному використанні в заданих умовах проявлялись достатньо рідко. Впевнитися, що ППЗ відповідає даній вимозі можна при його використанні шляхом тестування, а також при практичному застосуванні.

Так само як у техніці, ступінь надійності можна характеризувати ймовірністю роботи ППЗ без відмови протягом певного періоду часу. При оцінці ступеня надійності варто також ураховувати наслідки кожної відмови. Деякі помилки можуть викликати лише певні незручності при його застосуванні, тоді як інші помилки можуть мати катастрофічні наслідки. Тому для оцінки надійності ППЗ іноді використовують додаткові показники, що враховують вартість (шкоду) кожної відмови для користувача. Надійність визначається стійкістю та захищеністю.

Стійкість характеризується забезпеченням коректного функціонування, не зважаючи на задання неправильних (помилкових) вхідних даних, апаратну відмову й помилки під час виконання програми. У стійких системах реалізовано опрацювання невдач, що виникли, без втрати даних і припинення роботи.

Захищеність характеризується можливістю протистояти навмисним або ненавмисним деструктивним (руйнуючим) діям користувача та можливістю відновлювати визначений рівень працездатності й цілісність даних після таких дій.

На надійність функціонування ППЗ впливають насамперед фактори, що викликають збої або відмови при виконанні програми. Серед таких факторів можна виокремити: спотворення вхідних даних, що надходять від користувачів; відмови або збої в апаратурі ЕОМ; невиявлені помилки в ППЗ.

Спотворені дані в ряді випадків можуть бути причиною тільки помилок у результатах, які надаються користувачеві або накопичуються в пам’яті ЕОМ, і не впливають на надійність ППЗ. Однак деякі спотворені дані можуть виходити за область допустимих значень змінних. При цьому зростає ймовірність того, що спотворена величина буде оброблятися деякою послідовністю команд, що приводять або до відмови, або до збою функціонування.

Відмови або збої в апаратурі ЕОМ є значним чинником, що впливає на надійність функціонування програми. Деяка частина апаратних збоїв може приводити до спотворення виконання програм або до спотворення даних.

Невиявлені помилки в ППЗ є основними причинами ненадійності функціонування багатьох програм. Хоча в процесі налагодження основна частина помилок у програмах виявляється й усувається, завжди є ризик прояву якоїсь помилки при певному, раніше не випробуваному сполученні вхідних даних.

Функціональність –надання користувачу можливості виконувати набір функцій, які задовольняють завданням чи можливим потребам користувачів у відповідній предметній галузі, за допомогою ППЗ. Функціональність визначається завершеністю, коректністю, точністю та автономністю.

Завершеність характеризується ступенем притаманності ППЗ всіх необхідних частин і рис, що потрібні для виконання своїх явних і неявних функцій.

Коректна програма повертає правильний результат при поданні на вхід правильних даних і відповідає всім вимогам до специфікації даних.

Точність характеризується прийнятністю величини похибки в результатах, отриманих за допомогою ППЗ, з огляду на предметну галузь.

Автономність характеризується можливістю виконувати запропоновані функції без додаткового програмного забезпечення, що не входить в склад ОС або обладнання.

Зручність використання (практичність) – це характеристика ППЗ, згідно якої мінімізуються зусилля користувача щодо підготовки вхідних даних та аналізу отриманих результатів, а також викликаються позитивні емоції при його використанні. Зручність застосування визначається рівнем документованості та інформативності для користувача, комунікабельністю.

Рівень документованості та інформативності для користувача – властивість, що характеризує наявність, повноту, зрозумілість, доступність і наочність навчальної, інструктивної і довідкової документації, необхідної для застосування ППЗ та розуміння його призначення, прийнятих обмежень, форматів вхідних даних і результатів роботи окремих компонентів, а також поточного стану програми у процесі її функціонування.

Комунікабельність – властивість, що характеризується можливістю введення даних користувачем у зручній формі та отриманням користувачем корисних повідомлень в простій формі і з простим для розуміння змістом.

Ефективність (продуктивність) – це відношення рівня послуг, які отримує користувач при використанні ППЗ до об'єму використаних ресурсів комп'ютера. Ефективність визначається: ефективністю щодо часу, ефективністю щодо пам'яті, ефективністю щодо зовнішніх пристроїв.

Ефективність щодо часу – міра, що характеризується виконанням покладених на ППЗ функцій за прийнятний відрізок часу.

Ефективність щодо пам'яті – міра, що характеризується виконанням покладених на ППЗ функцій при певних обмеженнях на використовувану пам'ять.

Ефективність щодо зовнішніх пристроїв – міра, що характеризується виконанням покладених на ППЗ функцій при обмеженому наборі системних пристроїв.

Мобільність – збереження працездатності ППЗ при його переміщенні з одного апаратного і програмного оточення в інше без особливих зусиль. Мобільність визначається незалежністю від пристроїв, модульністю, сумісністю програмного забезпечення.

Незалежність від пристроїв – властивість, що характеризує працездатність програми на різноманітному апаратному забезпеченні.

Модульність – властивість, що характеризує програму із погляду організації її як системи модулів, причому зміна одного з них не впливає на інші модулі.

Сумісність програмного забезпечення – міра того, наскільки просто об'єднати різні програми разом для їх спільного застосування. Основи сумісності випливають із загальних проектних рішень. Наприклад, чи забезпечується засобами ППЗ, можливість зберігати та завантажувати дані, для їх подальшого опрацювання, з структурованих файлів даних які є добре описаними: CSV (Comma-Separated Values), INI (Initialization file), XML (eXtensible Markup Language).

Легкість у освоєнні тексту програми – це характеристика, за якою можна мінімізувати зусилля щодо вивчення й розуміння тексту програми і документації до неї. Легкість у освоєнні визначається документованістю для розробників, читабельністю, внутрішньою та зовнішньою погодженістю.

Документованість для розробників – властивість, що характеризує наявність документації для розробника, вимог до ППЗ, опису результатів різних етапів розробки, опису обмежень та їх обґрунтування, коментарів у тексті програми.

Читабельність – властивість, що характеризує легкість сприйняття тексту програм, можливість зрозуміти призначення кожного оператора та ідентифікатора.

Внутрішня погодженість повинна забезпечувати єдину термінологію, єдине трактування понять і значень. Особливе значення ця характеристика набуває при створенні програмних засобів, коли над проектом працює група фахівців, і в процесі роботи необхідні контакти для узгодження програмних модулів.

Зовнішня погодженість забезпечується однозначною відповідністю створюваного ППЗ вимогам, поданим у технічному проекті на його розробку.

Модифікованість – відображає можливість внесення змін у ППЗ без значних витрат часу на наступне налагодження. Ця характеристика містить у собі характеристику розширюваності, тобто характеризується можливістю розширення функціональних можливостей окремих компонентів ППЗ. Дана властивість не є істотною для малих проектів, але стає визначальною, коли відбувається створення великих програмних засобів.

1.4.3. Етапи проектування та створення педагогічного програмного комплексу навчання. Як було зазначено раніше ППЗ є однією з частин ППК навчання, який включає в також себе методичні та дидактичні матеріали і настанови, розраховані на використання відповідного ППЗ для комп’ютерної підтримки навчально-пізнавальної діяльності. Тому важливим є розробка методики застосування ППЗ у навчальному процесі. Технологія проектування та розробки такого ППК навчання складається з ряду етапів.

Перший етап. Визначення мети та цілей використання майбутнього ППК навчання.

Другий етап. Визначення вимог до майбутнього ППЗ, як складової ППК навчання. Пошук доцільних методів представлення та опрацювання даних в межах предметної галузі.

Третій етап. Добір змісту навчання відповідно до результатів першого й другого етапів. При цьому треба враховувати наступні фактори: встановлення міжпредметних зв'язків на рівні понять, формованих умінь і методів дослідження з метою створення інтегрованого змісту з окремих професійно-значимих питань; активне використання методів наукового дослідження і прийомів подання та опрацювання даних в конкретній предметній галузі; використання комплексу прийомів укрупнення дидактичних одиниць.

Четвертий етап. Структурування змісту, тобто виокремлення законів, понять та факторів, які необхідні для створення та використання ППЗ конкретного типу (навчального, контролюючого, демонстраційного).

На цьому етапі відбувається встановлення зв'язків між добраним змістом навчання і можливими методами навчання. Добір методів навчання залежить від таких елементів, як дидактичні цілі, зміст та організаційні форми навчання, характеристики комп'ютерної техніки. Окрім того добір методів навчання залежить від рівня вмінь учнів працювати з ПК і рівня "майстерності" викладача використовувати комп'ютер в навчальному процесі, оскільки комп'ютерне навчання, безсумнівно, вносить значні зміни в систему методів навчання.

П'ятий етап. Створення ППЗ – багатофункціональний процес, що поєднує безліч різнорідних робіт: розробку сценарію ППЗ, добір мови програмування, кодування, налагодження програми, тестування, документування.

Шостий етап. Розробка методики навчання з використанням ППЗ, тобто створення цілісного ППК навчання. Для реалізації цього етапу необхідне нагромадження значного фактичного матеріалу.

Сьомий етап. Проведення апробації з метою виявлення ефективності створеного ППЗ та ППК навчання вцілому.

Восьмий етап. Внесення змін та вдосконалення ППЗ та ППК вцілому за результатами проведеної апробації.

1.5. Науково-методичні підходи до навчання програмування

Як показує досвід, у процесі навчання програмування можна виокремити чотири стадії вміння і здатності студентів програмувати [58]: репродуктивне, осмислене, доказове, творче.

Репродуктивне програмування. Для набуття студентами компетентностей з програмування викладач за рахунок підібраної системи задач має організувати діяльність студентів по неодноразовому відтворенню викладених ним знань і показаних способів дій. Викладач дає завдання, а студенти їх виконують – розв'язують схожі завдання, працюють за інструкцією. Від того, наскільки завдання є важким, від здібностей студента залежить, як довго, скільки разів і з якими інтервалами студент має повторювати роботу. Відтворення та повторення розв'язування задач з програмування за завданнями викладача є головною ознакою стадії репродуктивного програмування. Сама назва характеризує лише діяльність студента, проте з опису можна побачити, що дана стадія включає організаційну діяльність викладача. Велика увага приділяється вдосконаленню способів інструктажу студентів. Окрім усних пояснень і показу прийомів розв'язування задач з програмування, для цієї цілі використовуються письмові інструкції, приклади частин програмного коду та ін.

Репродуктивне програмування характеризується меншою продуктивністю, в порівнянні з іншими стадіями, проте воно відіграє важливу роль. На основі цього виду програмування відбувається розв'язування задач знайомої структури «за зразком». Воно забезпечує розуміння нових прийомів програмування, застосування знань на практиці, якщо при цьому не потрібно їх істотне перетворення. Можливості репродуктивного програмування визначаються наявністю початкового мінімуму знань.

Репродуктивне програмування використовується на початковому етапі, коли від студента вимагається розв'язування задачі за зразком. Проте при появі задачі, яку студенту не вдається розв'язати раніше відомим йому способом, за зразком, у

студента відбувається виникнення «проблемної ситуації». Тобто відбувається перехід до інші стадії розв’язування задач, що забезпечує відкриття нових знань, формування нових підходів, які пізніше забезпечать йому можливість розв’язувати аналогічні задачі.

Осмислене програмування. На цій стадії навчання програмування від студента вимагається більше знань, ніж на стадії репродуктивного програмування, оскільки студенту необхідно самостійно визначити типи даних, необхідних для розв’язування задачі, вміння побудувати алгоритм, розуміння конструкцій мови програмування для написання тексту програми, можливість протестувати програму. Проте, на цій стадії навчання, студент ще може не відразу знаходити раціональний алгоритм розв’язування задачі.

Доказове програмування. ідея доказового програмування вперше була висловлена академіком А.П. Ершовим. Доказове програмування – це створення програм з доказом їх правильності [79]. Аналіз складності конструювання і доказ правильності алгоритмів і програм полягає в наступному. Для висновків про наявність помилок в алгоритмі чи програмі достатньо вказати тести, при виконанні яких відбудеться збій, відмова чи буде отримано невірний результат. Для твердження про “правильність” програми, необхідно довести, що правильна відповідь буде отримана для всіх допустимих вхідних даних. Таке твердження може бути доведене тільки шляхом вичерпного аналізу результатів виконання програми при будь-яких доступних даних. Проте, необхідно відмітити, що в педагогічних ВНЗ розгляд доказового програмування варто проводити лише в ознайомлювальному вигляді, оскільки воно має більше відношення до технічних спеціальностей.

Творче програмування. Творче програмування характеризується новизною свого результату для студента, своєрідністю процесу його отримання і істотним впливом на розумовий розвиток. Студенти, знаходячись на рівні творчого програмування, можуть забезпечувати самостійне розв’язання нових для них проблем, глибоке засвоєння знань, швидкий темп оволодіння новими знаннями, широту їх перенесення в відносно нові умови, здатність самостійно формулювати

задачі, використовувати навчальну, наукову, періодичну літературу з програмування в пошуках нових знань. В творчому програмуванні проявляються інтелектуальні здібності студента, його творчий потенціал. Основними критеріями творчого програмування є: швидкість – здатність генерувати велику кількість ідей в певну одиницю часу; гнучкість – здатність знаходити різні способи розв'язування однієї і тієї ж задачі; оригінальність – здатність знаходити нестандартні розв'язки, пропонувати нетривіальні ідеї; сприйнятливість – здатність до асоціативного мислення, реагування на окремі деталі чи протиріччя, здатність швидко переключатися з однієї ідеї на іншу.

Далі буде розглянуто ряд науково-методичних підходів, які можуть бути використані в процесі навчання програмування, як окремо один від одного так і в деякому сполученні.

1.5.1. Задачний підхід. Як і в більшості предметів природничо-математичного циклу, в програмуванні уміння та навички розв'язування задач формуються безпосередньо у процесі розв'язування задач на основі застосування знань відповідних теоретичних основ. Тому на сучасному етапі навчання програмуванню найпоширенішим є задачний підхід. М.М. Скаткін відзначає, що "сучасна психологія і дидактика розкрили значну роль задач у навчанні як одного з найважливіших чинників збудження і підвищення пізнавальної і практичної активності учнів" [65, с. 120].

У роботах Є.М. Кабанової-Мєллєр [105, с. 7], Г.С. Костюка [14] та ін. обґрунтовується концепція формування мислення, яку визначають як „задачний підхід". Розв'язування задачі розглядається як цілеспрямована діяльність, вагоме місце також відводиться розумовій діяльності, в ході якої і відбувається розвиток мислення студентів. Початковим етапом такого процесу найчастіше є проблемна ситуація, оскільки, найчастіше, мислити людина починає, коли в неї виникає потреба щось зрозуміти.

Відомий математик Д. Пойа розрізняє „чотири ступені у процесі розв'язування" задачі: розуміння постановки задачі; складання плану розв'язування задачі; здійснення плану; аналіз результатів [178]. При цьому

задача повинна бути поставлена коректно, щоб для її розв’язування був потрібен набір визначених, обмежених у кількості і принципово доступних даних. При цьому неістотно, чи є ці дані в наявному матеріалі, чи вказується, як можна їх отримати.

Відзначимо, що в програмуванні, як і в математиці, під методом розв’язування задач розуміють сукупність прийомів розумової діяльності або логічних дій і операцій, за допомогою яких розв’язується широкий клас задач, а під способом розв’язування задач розуміють сукупність прийомів розумової діяльності або логічних дій і операцій для розв’язування окремої задачі. У програмуванні, як і в математиці, практично кожна задача має декілька способів (алгоритмів) розв’язування.

Особливістю розв’язування задач з програмування є те, що розв’язування задачі з програмування, як правило, передбачає реалізацію алгоритму деякою конкретною мовою програмування, що вимагає від студента не лише знання відповідних мовних конструкцій, а і вміння працювати в певному середовищі програмування.

Можна зазначити, що розв’язування задачі з програмування може складатися з трьох частин: побудова інформаційної моделі задачі, пошук і побудова алгоритму її розв’язування та реалізація цього алгоритму конкретною мовою програмування. Причому для задачного підходу перша з них є більш вагомою. Тобто можна зазначити, що мова програмування є інструментом, за допомогою якого розв’язується задача і який можна поміняти, якщо в цьому виникає необхідність.

При розв’язуванні задач на початковому етапі вивчення програмування, педагог має погоджуватися з будь-який правильним алгоритмом, що був запропонований студентом для розв’язування задачі, якщо даний алгоритм буде приводити до отримання правильного результату. На наступних етапах викладач разом з студентом чи навіть з всією групою студентів має проводити аналіз ефективності запропонованих алгоритмів з погляду використаних конструкцій

мови програмування, кількості операцій, часової складності, кількості проміжних змінних і т.п. і в результаті вибрати найкращий з запропонованих алгоритмів.

Такий підхід сприяє підвищенню інтересу студентів до навчання, надає їм впевненості у своїх можливостях і знаннях, що, у свою чергу, активізує їх пізнавальну активність, стимулює навчальну діяльність студентів. В той же час розв'язування задачі за готовим алгоритмом, запропонованим викладачем, не вимагає від студента самостійного мислення. А набуття вмінь та навичок розв'язувати задачі відбувається тільки в результаті самостійної праці.

В циклі лабораторних робіт доцільно застосовувати задачі за рівнями складності в межах кожної теми. При цьому доцільно усі задачі поділити на три рівні складності. Задачі першого рівня повинні передбачати засвоєння студентом навчального матеріалу з програмування на базовому рівні. Задачі другого рівня повинні бути більш трудомісткі, вимагати вдумливості, уміння аналізувати. Нарешті, задачі третього рівня – найскладніші. Наприклад, задачі рівня шкільних олімпіад. Для розв'язування цієї категорії задач від студента потрібне застосування творчого підходу, розвинене алгоритмічне мислення, зацікавленість процесом програмування. Такий підхід забезпечує індивідуальний шлях для кожного студента при навчанні програмування, надає можливість точніше визначити рівень знань кожного студента [90].

1.5.2. ітераційно-поступальний підхід. Досить часто методика навчальної діяльності являє собою ітераційний процес. Це дає можливість запропонувати ітераційно-поступальний підхід для навчання програмування.

Розглядаючи ітерацію, як циклічне наближення до певної мети, можна застосувати ітераційний підхід, як при поданні лекційного матеріалу, так і в процесі виконання лабораторних робіт. Тим більше, що специфіка задач, призначених для виконання на практичних та лабораторних роботах, досить часто відповідає такому ітераційно-поступальному процесу, який полягає в модифікації раніше створеної моделі згідно нових методів та підходів, що були отримані студентами на лекційних заняттях. Таким чином побудова кінцевої моделі представляє собою ітераційний процес, на кожному кроці якого відбуваються

певні зміни, згідно набутих компетентностей. Власне це і дозволяє застосовувати ітераційний підхід навчання в процесі навчання програмування [91].

Для прикладу, розглянемо послідовність вивчення методів сортування при навчанні програмування. На початковому етапі після вивчення лінійних масивів, студентам демонструються і пояснюються методи сортування, які базуються на вже отриманих компетентностях застосування умовного оператору та циклів. Такими методами можуть бути: метод бульбашки, метод прямої вставки, методом вибору. В подальшому після вивчення рекурсії, викладач знову може повернутися до розгляду сортування лінійних масивів, проте з застосуванням отриманих компетентностей щодо рекурсії в програмуванні і показати методи сортування, серед яких можуть бути: сортування злиттям, швидке та пірамідальне сортування.

1.5.3. Підхід парного програмування. У процесі навчання програмування доцільно використовувати підхід “парного програмування”, суть якого полягає в тому, що під час виконання завдань два студента одночасно працюють за одним комп’ютером. При цьому один із студентів набирає код в той час як другий студент зосереджений на задачі в цілому і переглядає код, що набирається першим студентом. В цьому випадку студенти, що працюють в парі, постійно спілкуються з приводу завдання, здійснюють взаємонавчання і взаємоконтроль.

Існує декілька стилів розподілу ролей при парному програмуванні:

- *ведучий-ведений*. Використовується, коли ставиться за мету навчити менш досвідченого студента за підтримки більш досвідченого. В такому випадку більш досвідчений студент підказує, направляє, дає рекомендації менш досвідченому, який в безпосередньо пише код;
- *на рівних*. В цьому випадку студенти працюють однаково, час від часу змінюючи один одного;
- *водій-штурман*. В цьому випадку студенти отримують різні ролі. Один пише код і розбирається в деталях, а інший займається архітектурою коду і розв’язанням підзадач;

- *пінг-понг*. Застосовується на стадії тестування, коли один студент формулює тест для певних вхідних даних, а інший – перевіряє правильність виконання і достовірність отриманого результату.

Для всіх стилів розподілу ролей при парному програмуванню, окрім стилю ведучий-ведений, обов'язковою є умова обміну ролями студентів в процесі роботи, тобто періодично студенти, що працюють в парі, повинні мінятися ролями за робочим місцем (комп'ютером).

Серед позитивних сторін даного методу можна виокремити наступні: студенти навчаються не лише програмувати, а і колективно вирішувати проблеми, обговорювати проблеми, знаходити компроміси; працюючи в парі, студенти вчаться один в одного; навіть найкращий студент не може знати всього, тобто наявність двох студентів дозволяє швидше і ефективніше розв'язувати задачі; студенти краще концентруються і менше відволікаються на сторонні справи; зменшення кількості помилок, які виникають процесі роботи; обидва студенти будуть знати і розуміти весь програмний код, ніж якби вони писали лише свою частину, як це буває при колективному підході, в результаті чого при необхідності вони зможуть більш ефективно вносити зміни.

Програмування в парі може бути набагато результативнішим і цікавішим, ніж програмування поодинці, якщо організоване правильно. і навпаки, може бути жахливим і нудним у порівнянні з роботою поодинці, якщо – неправильно. Значна частина спроб парного програмування губиться в наступних випадках:

- В парі присутній студент добре досвідчений в програмуванні. В такому випадку питання менш досвідченого студента про розроблюваний код часто залишається без відповіді. Окрім того досвідчений в програмуванні студент намагається сам розв'язувати поставлену задачу, не допускаючи напарника до написання коду, а у разі передачі повноважень написання коду напарнику сам втрачає інтерес до розв'язання задачі.
- Один з студентів досить часто займає жорстку ультимативну позицію з приводу всіх рішень, які стосуються поставлених задач. У такому випадку не може йти мова про взаємну допомогу та навчання в парі.

- Один з студентів самостійно пише код і в процесі написання і в процесі написання відволікає напарника сторонніми справами. Це порушує базову ідею про взаємну залученість напарників у процес.
- Напарники не спілкуються один з одним, не коментують свої дії і рішення в процесі роботи. За відсутності зворотного зв’язку сенс пари втрачається.
- Студенти в парі паралельно працюють за двома комп’ютерами.

1.5.4. Підхід колективного програмування. В більшості випадків сучасне програмування є колективним, і вклад кожного окремого програміста в спільно створюваний ПЗ впливає на успіх всієї команди. Необхідно розуміти важливість формування і розвитку у студентів таких важливих якостей, як уміння працювати в колективі і вміння співпрацювати з іншими колективами, наприклад колективом вчителів-предметників з інших дисциплін чи колективом методистів.

Учасники команди працюють над спільною задачею, шукають шляхи її розв’язання. Кожен сумлінний учасник несе особисту відповідальність за код програми і вважає доцільним знаходити і відстоювати кращі розв’язки проблеми. Такий вид навчальної діяльності також спонукує студентів до самостійної творчої роботи і глибокого усвідомленого вивчення курсу програмування.

Однією з основних проблем колективної розробки є розподіл праці – від рівноправних співвиконавців до організації в вигляді жорсткої ієрархії. Тому можна виокремити такі технічні командні ролі.

Рівноправні співвиконавці. В рамках колективних засідань студенти проводять колективне обговорення ПЗ, погоджують його розбиття на підзадачі та їх розподіл між виконавцями, розв’язують отримані підзадачі, а потім формують результуючий ПЗ. В свою чергу при рівноправному співвиконанні може бути два підходи: перший – коли кожен студент отримує власну підзадачу займається лише нею і не втручається в розв’язання інших підзадач; другий – коли в процесі розв’язування відповідальність за розв’язування підзадач може бути перерозподілена в зв’язку з вивільненням часу чи недостатності знань окремих виконавців.

Колектив головного програміста. В колективі є головний розробник, яким виступає один із студентів, а всі інші студенти, що входять до колективу, допомагають йому в розробці ПЗ.

На нашу думку при навчанні програмування та подальшої підготовки майбутніх учителів до розробки ППЗ одним з базових підходів має виступати задачний підхід, в основу якого покладена цілеспрямована практична діяльність. Проте задачний підхід не має бути єдиним підходом при навчанні програмування. В практичній діяльності навчання програмуванню, викладач має використовувати не якийсь окремий методичний підхід, а злиття декількох підходів в один для більш ефективного навчання.

Висновки до розділу 1

1. Навчання програмування в педагогічних ВНЗ дещо відстає від сучасних технологій створення ПЗ, а отже і ППЗ. Актуальною проблемою є підготовка майбутніх учителів математики та інформатики до навчання концептуальних засад створення та проектування ПЗ, що можуть бути використані і при створенні ППЗ, на основі сучасних мов та середовищ розробки програмного забезпечення.
2. Навчити програмувати студентів можна тільки за умови розробки ними нетривіальних програмних засобів, якими у педагогічних ВНЗ можуть бути ППЗ.
3. Навчання програмування варто починати з застосування процедурної мови програмування, в подальшому розширюючи цю мову об’єктно-орієнтовними можливостями.
4. Для більш ефективного навчання програмування та створення ППЗ в педагогічних ВНЗ доцільно застосовувати поєднання різних науково-методичних підходів, зокрема: задачного, ітераційно поступального, парно-програмувального, колективного. Базовим підходом, може виступати задачний підхід, оскільки компетентності з програмування формуються безпосередньо у процесі розв’язування задач на основі застосування набутих теоретичних знань.

5. При формулюванні методичних вимог до ППЗ потрібно враховувати специфіку конкретної предметної галузі і відповідного їй навчального предмету. Визначаючи дидактичні вимоги, яким повинні задовольняти ППЗ, необхідно враховувати також особливості добору змістового наповнення ППЗ, аргументованого певними методичними цілями, і забезпечувати перевірку педагогічної ефективності використання ППЗ.
6. Підготовку до розроблення ППЗ доцільно проводити базуючись на об’єктно-орієнтованій парадигмі, яка буде базуватися на об’єктно-орієнтованій технології програмування в сукупності з окремими елементами структурної та модульної технології. Оскільки використання саме об’єктно-орієнтованої парадигми та об’єктно-орієнтованої технології програмування забезпечує отримання ПЗ, які краще моделюють предметну галузь, а змодельовані системи простіше адаптувати до нових умов та легше модернізувати.
7. Доцільним є використання сучасних вільно поширюваних мов та середовищ програмування, зокрема Free Pascal та Lazarus, використання яких надає можливість створювати вільно поширювані програмні засоби та надає поштовх до формування певних етичних і правових норм поведінки стосовно використання засобів комп’ютерних технологій.

Основні результати першого розділу опубліковані в роботах [51; 54; 56; 122; 123; 125; 126; 127; 130]

РОЗДІЛ 2
КОМПОНЕНТИ МЕТОДИЧНОЇ СИСТЕМИ ПІДГОТОВКИ МАЙБУТНІХ УЧИТЕЛІВ МАТЕМАТИКИ ТА ІНФОРМАТИКИ ДО РОЗРОБЛЕННЯ ПЕДАГОГІЧНИХ ПРОГРАМНИХ ЗАСОБІВ

2.1. Загальна методика дослідження проблеми

Актуальність навчання майбутніх учителів математики та інформатики розробляти ППЗ, важливість впровадження такого навчання для студентів педагогічних ВНЗ потребує переосмислення теоретико-методологічних засад навчання програмування в цілому та концептуальних підходів його здійснення.

Провідною ідеєю дослідження є положення про те, що підготовка до розроблення ППЗ на основі використання вихідних кодів реально використовуваних ППЗ для майбутніх учителів математики та інформатики забезпечує відповідне професійне спрямування, забезпечує належний рівень міжпредметних змістових зв'язків навчального матеріалу.

Концепція дослідження ґрунтується на використанні об'єктно-орієнтованої парадигми програмування засобами середовища програмування Lazarus для створення авторської комп'ютерно орієнтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення ППЗ.

Провідні ідеї дослідження відбиті у ***гіпотезі***, яка ґрунтується на припущенні, що цілеспрямоване використання науково обґрунтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів, що базується на використанні вихідних кодів реально використовуваних ППЗ, буде сприяти: активізації пізнавальної діяльності студентів; набуттю практичних навичок створення реальних ППЗ; опануванню особливостей аналізу і опрацювання математичних об'єктів; покращенню міжпредметних зв'язків, змістовної єдність програмування з іншими навчальними предметами, для яких будуть створюватися ППЗ.

Методологічною основою дослідження є теорія пізнання й відображення дійсності в людській свідомості; діалектичний принцип взаємозв'язку й взаємообумовленості закономірностей і явищ об'єктивної дійсності;

взаємозв’язок теорії та практики у процесі навчання; активна роль особистості у пізнанні й перетворенні навчального матеріалу; системний підхід до теоретичного осмислення та практичної організації процесу навчання інформативним дисциплінам; теоретичні основи побудови алгоритмів та програм; концепції сучасних парадигм програмування; системно-структурний підхід до аналізу навчальної діяльності; положення психології та педагогіки про активність особистості у процесі навчання; теорія навчання у вищий школі; загальнодидактичні положення; основні положення щодо формування системи та структури організації навчання.

Для досягнення мети і реалізації завдань дослідження застосовувався комплекс методів, серед яких можна вмокремити: аналіз, систематизація, узагальнення наукової вітчизняної та зарубіжної фахової, педагогічної та науково-методичної літератури; педагогічне прогнозування і моделювання для побудови ефективного навчально-виховного процесу, згідно з предметом дослідження; спостереження за навчальною діяльністю студентів педагогічних ВНЗ; бесіди й опитування вчителів для ґрунтовного вивчення досвіду їх професійної діяльності; педагогічний експеримент з метою експериментальної перевірки ефективності розроблених компонентів методичної системи; підхід експертного оцінювання ППЗ для отримання оцінки результатів навчання в процесі педагогічного експерименту; математично-статистичний метод, за допомогою якого визначено кількісні залежності між показниками, отриманими в результаті педагогічного експерименту та проведено якісний аналіз цих кількісних залежностей.

Дослідження проводилося протягом 2006–2013 років, воно охоплювало три етапи науково-педагогічного пошуку. Перший етап полягав у збиранні та аналізі даних для виявлення наукової проблеми стосовно підготовки майбутніх учителів до розроблення ППЗ та висуненні гіпотези дослідження. Другий етап полягав у збиранні та аналізі даних для уточнення гіпотези та побудови теоретичної моделі дослідження. Третій етап – полягав в емпіричній перевірці побудованої теоретичної моделі, обґрунтуванні вірогідності результатів проведеного

експерименту та здійсненні впровадження основних результатів наукової роботи в практику.

Вірогідність результатів дослідження забезпечено обґрунтованістю вихідних положень щодо підготовки майбутніх учителів до розроблення ППЗ; відповідністю комплексу взаємодоповнюючих методів дослідження меті, об'єкту, предмету і завданням дослідження; вивченням об'єкта дослідження з урахуванням усіх вимог щодо надійності, обґрунтованості, точності пізнавальних процедур; поєднанням кількісного і якісного аналізів теоретичного та емпіричного матеріалів; застосуванням елементів математичної статистики для опрацювання експериментальних даних.

2.2. Методи навчання створення педагогічних програмних засобів

Український педагогічний словник дає визначення поняттю «метод», наступним чином: "Метод – спосіб організації практичного й теоретичного освоєння дійсності, зумовлений властивостями об'єкту дослідження. З розвитком науки відбувається розвиток і диференціація методів, що приводить до виникнення вчення про методи – методології" [232, с. 205]. У більш широкому розумінні метод – спосіб досягнення мети, сукупність прийомів та операцій теоретичного або практичного освоєння дійсності, а також людської діяльності організованої певним чином [38, с. 633].

Для досягнення освітніх цілей застосовуються методи навчання. Методи навчання (від гр. mthodos – спосіб пізнання, шлях дослідження) – це упорядковані способи діяльності викладача та студента, що спрямовані на засвоєння студентом системи компетентностей, їх виховання і загальний розвиток [140, с. 215]. У методах навчання виокремлюють два види: метод навчання як інструмент діяльності викладача для виконання навчальної функції – навчання; а також метод навчання як спосіб пізнавальної діяльності студентів з оволодіння компетентностями – учіння.

Методи навчання в широкому сенсі у відношенні до всіх навчальних предметів складають предмет дослідження дидактики. В завдання методик навчання входить розробка застосування вже досліджених дидактикою загальних

методів навчання з урахуванням особливостей змісту і наукових методів дослідження тієї чи іншої науки.

Відомо, що на вибір методів навчання впливають такі чинники, як специфіка дисципліни, цілі навчання, зміст курсу, складність навчального матеріалу, кількість відведеного часу (кредитів) на дисципліну, рівень підготовленості студентів, рівень мотивації навчання студентів, матеріально-технічна база навчального закладу. Тому в процесі навчання створення ППЗ доцільно використовувати методи навчання, які обумовлюються видами навчально-пізнавальної діяльності студентів [62; 205; 211; 229].

Серед методів навчання важливе місце займає метод активного навчання. Метод активного навчання – це сукупність педагогічних дій і прийомів, спрямованих на організацію навчального процесу, створення спеціальними засобами умов, які мотивуватимуть студента до самостійного, ініціативного і творчого засвоєння навчального матеріалу в процесі пізнавальної діяльності [166]. Даний метод впливає на активізацію особистих здібностей студентів. М. Новік визначає наступні характерні особливості активного навчання [166]: примусова активізація мислення, коли студент змушений бути активним незалежно від його бажання; тривале залучення студентів у навчальний процес, оскільки їх активність повинна бути не короткочасною і епізодичною, а в значній мірі стійкою і тривалою (тобто протягом всього курсу); самостійна, творча робота щодо отримання рішень, підвищений ступінь мотивації та емоційності студента.

Проблемам розробки та використання методів навчання присвячені численні дослідження. Класифікації методів навчання відрізняються одна від одної ознакою, покладеною в основу кожної з них. Так, поширені класифікації методів за джерелами знань; за характером логіки пізнання; за рівнем проблемності засвоєння знань; на основі цілісного підходу до процесу навчання; на основі цілісного підходу до навчальної діяльності та інші.

Спільним для всіх підходів є те, що в кожному з них відображаються три групи ознак, які характеризують: навчально-пізнавальну діяльність; педагогічну діяльність; предмет спільної діяльності учасників педагогічного процесу. Тому

для того щоб обрати, задати або описати практично реалізований за тих або інших умов метод навчання, потрібно вказати ознаки, що належать усім трьом групам. Різноманіття можливих методів навчання – це різноманіття варіантів добору ознак, що належать цим групам.

Розглянемо класифікації методів навчання з точки зору застосування цих методів при навчанні дисципліни “Технології програмування та створення ППЗ”. Серед різних класифікацій можна виокремити три: за способом передачі відомостей від викладача до студента; за основними видами дидактичних проблем, що вирішуються на заняттях; за характером пізнавальної діяльності студентів.

І. За способом передавання відомостей від викладача до студента визначають такі методи [9; 59; 160]: словесні або вербальні (бесіда, пояснення, розповідь, лекція, інструктаж), наочні або демонстраційні (ілюстрування та демонстрування), практичні (вправи, лабораторні, практичні та дослідні роботи).

Під час підготовки до створення ППЗ словесні методи використовуються при викладанні лекційного матеріалу: студентам наводяться поняття парадигм та технологій програмування, що можуть бути використані при створенні ППЗ; висувається комплекс вимог до ППЗ та показники якості ППЗ. У наочних методах головну роль відіграє демонстрація та візуалізація, так викладач демонструючи вихідний код вже готового ППЗ, на розгляді якого базується підготовка до розроблення ППЗ, в наочній формі може показати суттєві деталі. Практичні методи можуть застосовуватися на практичних роботах е вигляді покрокового виконання програмного коду демонстраційного ППЗ, для більш наочного показу тих чи інших реалізованих характеристик ППЗ, з внесенням часткових змін в вихідний код; на лабораторних роботах при розв’язуванні завдань, які будуть базуватися на створенні деякого ППЗ з “обмеженими” набором характеристик.

При підготовці до створення ППЗ наголос робиться на практичні методи, оскільки під час їх застосування студенти отримують не лише нові знання, але й практичні навички. Роль викладача полягає в постановці цілей роботи та контролі за її виконанням. В діяльності студентів переважає практична робота, під час якої

особливу роль відіграє самостійний процес мислення, що надає можливість здійснити пошук підходів до розв'язування задач, вибір технологій створення ППЗ, перегляд стандартних алгоритмів та добір потрібного з них.

II. Класифікація методів навчання за характером пізнавальної діяльності студентів [9; 118; 144; 160]:

Інформаційно-рецептивний (пояснювально-ілюстративний) метод навчання створення ППЗ застосовується найчастіше при поданні теоретичного матеріалу на лекціях та практичних заняттях і в ході формулювання задач для лабораторних робіт. Викладач пояснює матеріал, студенти його сприймають. При навчанні створення ППЗ даний метод може використовується при введенні понять, розгляді технологій створення ППЗ, підходів та етапів розробки ППЗ та розгляді середовищ мов програмування. При використанні цього методу діяльність студентів зводиться до усвідомленого сприйняття і запам'ятовування навчального матеріалу. Інформаційно-рецептивний метод вимагає застосування словесних (розповідь, читання навчальної і наукової літератури), наочних (блок-схеми, графіки, дерева класів), технічних (комп'ютер) засобів навчання.

Репродуктивний метод навчання створення ППЗ впливає на формування у студентів умінь використовувати здобуті знання, відтворенні знань і способів дій, діяльності за алгоритмом. Використання даного методу здійснюється, як правило, через систему доцільно дібраних завдань, наприклад, завдань на: перевірку знань технологій створення ППЗ; перевірку знань підходів та етапів розробки ППЗ; пояснення сутності операцій та методів класів; застосування належним чином вимог та показників щодо якості ППЗ; перевірку вмінь працювати в інтегрованому середовищі мови програмування; побудову класів, дерев класів та написання програми за зразком; використання і адаптацію стандартних алгоритмів; пояснення фрагментів програм; аналіз складності алгоритмів (за часом та пам'яттю); формування навиків тестування програми.

Завдання викладача при використанні репродуктивного методу навчання полягає в формуванні завдань, що виконуються студентом, та допомозі у виправленні допущених помилок.

Метод проблемного навчання створення ППЗ полягає в тому, що викладач висуває проблему, поясняючи навчальний матеріал, та при необхідності допомагає студентам у розв'язуванні цієї проблеми, демонструючи шляхи її розв'язання, керує процесом систематизації і закріплення здобутих знань. Проблемне подання навчального матеріалу сприяє активізації процесу навчання, підвищенню уваги і пізнавальної активності студентів. Цей метод навчання використовується під час лекцій у формі проблемного викладу навчального матеріалу викладачем, навчальної діяльності студентів під час лабораторних занять, самостійного дослідження навчального завдання. При навчанні створення ППЗ, даний метод може використовуватися при розгляд різних підходів та методів реалізації тих чи інших характеристи розроблюваного ППЗ і вибору найбільш прийнятного, аналізу класів, що проектуються та необхідної кількості їх характеристик з погляду розробника та користувача, побудові вимог з недостатніми чи надлишковими характеристиками розроблюваного ППЗ.

Евристичний (частково-пошуковий) метод навчання створення ППЗ – це метод організації пошукової, творчої діяльності на основі теорії поелементного засвоєння знань і способів діяльності. Система спеціально розроблених навчальних задач допомагає студенту самостійно виконувати кожен з етапів розв'язання конкретної задачі. Найбільш виразною формою евристичного методу є евристична бесіда, яка складається із серії взаємопов'язаних запитань, кожне з яких є кроком на шляху розв'язання задачі і які потребують від студента здійснення невеликого пошуку. При навчанні створення ППЗ, даний метод може використовуватися при побудові інформаційних моделей, побудові дерев класів та виборі раціональних підходів до реалізації тих чи інших характеристик розроблюваного ППЗ.

Дослідницький метод навчання створення ППЗ передбачає самостійний пошук студентами розв'язку поставленої задачі, тобто відбувається пошукова творча діяльність студентів стосовно розв'язування нових для них задач. При навчанні створення ППЗ, даний метод може використовуватися при аналізі предметної галузі, в межах якої буде створюватися ППЗ, побудові математичної

моделі задачі, виокремленні окремих об’єктів та визначення в них особливих характеристик.

Метод інтерактивного навчання створення ППЗ доцільно застосовувати при проведенні лабораторних занять. Інтерактивна діяльність при цьому передбачає організацію і розвиток спілкування студентів у процесі спільного розв’язування навчальних завдань. Інтерактив виключає домінування як однієї особи над іншими учасниками дискусії, так і однієї думки над іншою. Даний метод при навчанні створення ППЗ може застосовуватися у вигляді парно-програмувального чи колективного підходу до створення ППЗ чи його окремих елементів.

Інформаційно-рецептивний і репродуктивний методи забезпечують засвоєння студентами передбачених робочою програмою фактів, методів, алгоритмів, тощо і формування у них вмінь і навичок їх застосування, що є передумовою успішного навчання і творчої діяльності студентів, тоді як методи проблемного навчання, евристичний метод, дослідницький метод – формують риси творчої особистості студентів.

На нашу думку при підготовці до створення ППЗ, викладач має застосовувати не якийсь окремий метод навчання, а поєднувати та використовувати цілий набір методів для більш якісного і успішного навчання.

2.2.1. Рівнева диференціація навчання

Диференціація (з латинської "difference") означає розділення, розшарування цілого на різні частини, форми, ступені.

Диференціація вищої освіти – один з основоположних принципів формування особистості. У її основі лежить необхідність врахування індивідуальних особливостей студентів та забезпечення можливості кожному студенту вибирати різні форми навчання. Для цього повинна бути створена гнучка, адекватна запитам студентів система освіти [138, с. 14].

У вітчизняній та зарубіжній психолого-педагогічній науці проблема диференціації навчання з урахуванням здібностей учнів досліджувалась у різних аспектах: вивчення рівнів, форм і видів диференціації; наведення психологічних аспектів проблеми диференціації; дослідження диференціації навчання як засобу

запобігання неуспішності; вивчення пізнавальної активності студентів; розвиток пізнавальних можливостей, формування здібностей студентів.

В педагогічній і психологічній літературі не існує єдиного загальноприйнятого визначення поняття "диференціації навчання". У працях дидактиків Ю. К. Бабанського, Н. К. Гончарова, Н. М. Шахмаева диференціація розглядається як особлива форма організації навчання з урахуванням типологічних індивідуально-психологічних особливостей учнів і особливостей взаємозв'язку вчителя – учнів [98]. На основі узагальнення психолого-педагогічних і методичних досліджень розроблена концепція диференціації навчання, в якій підкреслюється, що "Диференціація навчання виступає як визначальний фактор демократизації та гуманізації системи освіти" [138, с. 13].

В психолого-педагогічній літературі розглядаються різні аспекти проблеми диференціації навчання та умови її здійснення. Більшість дослідників підкреслюють роль диференціації навчання в навчальній діяльності як чинника формування й розвитку пізнавальної активності студентів. У роботах інших дослідників диференціація навчання пов'язується з такою організацією навчального процесу, для якої характерно варіювання змісту, методів або темпу навчання, здійснюваного з урахуванням індивідуальних особливостей студентів.

Дослідниками [26; 138; 152] було визначено дві форми диференціації: рівневу (внутрішню), як сукупність прийомів та засобів навчання, що використовуються для забезпечення досягнення студентами різного рівня знань на основі врахування індивідуальних можливостей; профільну (зовнішню), яка передбачає навчання різних груп студентів за різними програмами та планами, що відрізняються змістом, структурою, обсягом вимог до компетентностей.

Аналіз педагогічної практики показує, що основна увага у вищих навчальних закладах зосереджується на рівневому диференціюванні навчання. Його основна особливість полягає в диференціації вимог до компетентностей студентів: явно з'ясовується рівень обов'язкової підготовки, який задає нижню межу засвоєння матеріалу. Цей рівень має бути доступним і посильним кожному. На його основі формуються підвищені рівні оволодіння дисципліною. Студенти отримують

право і можливість, навчаючись в одній групі і за однією програмою, вибирати той рівень засвоєння, що відповідає їх потребам, інтересам, здібностям, але не нижче мінімального.

Таким чином, обов'язком студента стає виконання обов'язкових вимог, що надає йому можливість мати позитивну оцінку з дисципліни. Це кардинально змінює традиційні підходи до організації навчання: не слід вирішувати за студента, який рівень засвоєння відповідає його здібностям, але слід створити у процесі навчання такі умови, при яких досягнення обов'язкового рівня буде реальним, студенти, здатні рухатися далі, будуть зацікавлені в цьому просуванні.

Рівні засвоєння висуваються студентам у формі переліку компетентностей, які вони повинні набути, зразків завдань, які повинні навчитися вирішувати. Але і при цій формі диференціації пояснення для всіх студентів даються знову ж таки на одному, частіше середньому або підвищеному рівні. Отже, особливе значення для впровадження в практику будь-яких форм і прийомів диференційованого навчання має організація предметного змісту навчального матеріалу. Центральне місце в ньому відводиться системам завдань, оскільки вони служать основними засобами формування прийомів навчальної діяльності студентів.

Концептуальними положеннями рівневої диференціації є: обов'язковий рівень повинен бути заданий по можливості однозначно, у формі, що не допускає різночитань, двозначностей; будучи основним робочим механізмом, обов'язковий рівень повинен забезпечувати гнучкість навчання і його адаптивність, можливості для еволюційного розвитку студента; виокремлення і відкрите пред'явлення студентам результатів навчальних досягнення (за рівнями); формування опорних знань: в усіх студентів групи незалежно від їх здібностей і навчальних можливостей повинні бути сформовані опорні знання та вміння; послідовність у просуванні за рівнями навчання; відповідність між змістом, контролем та оцінкою; добровільність у виборі рівня навчання: кожен студент добровільно вибирає рівень засвоєння навчального матеріалу; мотивація, а не констатація; нова психологічна установка для студента: "візьми стільки, скільки можеш, але не нижче обов'язкового"; студент повинен відчувати навчальний успіх.

У зв’язку з цим рівнева диференціація навчання передбачає наявність базового обов’язкового рівня підготовки, якого зобов’язаний досягти студент. Обов’язковий рівень виступає основою для диференціації та індивідуалізації вимог до студентів та повинен бути реально доступний для всіх студентів. Поряд з обов’язковим рівнем студенту надають можливість підвищеної підготовки, яка повинна визначатися глибиною оволодіння змістом навчального предмета.

Суттєвою особливістю технології рівневої диференціації навчання є її органічний зв’язок з системою контролю результатів навчального процесу та системою оцінювання досягнень. Альтернативою традиційному способу оцінки "вирахуванням" є "оцінка методом складання", в основу якої покладається обов’язковий, мінімальний, рівень підготовки, досягнути який обов’язково повинен кожний студент. Критерії більш високих рівнів будуються на базі урахування того, що досягнуто понад обов’язковий рівень, і системи заліків.

В основі нашого дослідження була покладена технологія рівневої диференціації навчання на основі обов’язкових результатів, за якою пропонується введення двох рівнів: обов’язкового рівня підготовки (рівень, якого повинен досягти кожен) та підвищеного рівня підготовки (рівень, який повинен забезпечити студенту, що цікавиться навчальним предметом, можливість проявити свої здібності). Простір між рівнями обов’язкової і підвищеної підготовки заповнено своєрідними "сходами" діяльності, добровільне сходження якими від обов’язкового до підвищеного рівня дозволяє реально забезпечити студенту постійне перебування в зоні найближчого розвитку навчання на індивідуальному максимально посильному рівні [72].

Враховуючи можливість кожним окремим студентом добровільно обирати рівень підготовки своїх навчальних досягнень, сприяє психологічному комфорту кожного студента, формує почуття поваги до себе і до оточуючих, виховує відповідальність і здатність до прийняття рішень. Практичне здійснення рівневої диференціації навчання не повинно означати, що одним пропонується більший обсяг матеріалу, а іншим менший. Кожен повинен пройти через повноцінний навчальний процес, який ні для кого не може бути обмежений вимогами

мінімуму. Інакше й рівень обов’язкової підготовки не буде досягнутий, і компетентності студентів, потенційно здатних на більше, можуть бути втрачені.

2.3. Пропедевтика навчання студентів створення ППЗ

Перш ніж створювати ППЗ, студент повинен оволодіти основами алгоритмізації і ООП, розуміти призначення ППЗ та бути обізнаним у предметній галузі, в межах якої буде розроблюватися ППЗ. Тому пропедевтикою підготовки до створення ППЗ можуть виступати такі дисципліни та їх окремі питання згідно галузевих стандартів вищої освіти з підготовки бакалаврів за напрямом 6.04.02.01 “Математика” та 6.04.03.02 “Інформатика*”:

- окремі питання дисципліни “Інформатика” (ПМ.04.02), при розгляді яких буде відбуватися навчання практичного застосування вже створених ППЗ та середовищ професійного призначення. Зокрема такими питаннями є: ПМ.04.02.11 – “діяльнісні середовища професійного призначення (професійні математичні та фізичні пакети) і їх використання в навчальному процесі”; ПМ.04.02.12 – “програмні засоби навчального призначення для підтримки вивчення математики в школі та ВНЗ“; ПМ.04.02.13 – “технології розв’язування задач з використанням засобів сучасних інформаційних технологій”;
- дисципліна “Моделювання” (ПМ.04.03) чи окремі питання дисципліни “Інформатика”, що стосуються моделювання, в ході навчання яких буде розкрито поняття про модель та моделювання, розглянуті різні види моделювання в природничих і технічних науках;
- окремі питання дисципліни “Алгоритмізація і програмування” (ПМ.04.04), в межах яких будуть розглянуті основні питання побудови моделей задач, складання і реалізація алгоритмів мовою програмування, розв’язування практичних задач засобами мови програмування;
- питання дисципліни “Алгоритмізація і програмування”, в межах яких вивчається “Об’єктно-орієнтоване програмування” (ПМ.04.04.05), оскільки ООП є інструментом, перш за все, для створення великих

програм, і отже і ППЗ. Зокрема переваги використання ООП були розглянуті в параграфі 1.3.

Розглянемо більш докладніше, як саме вивчення ООП може виступати пропедевтичним курсом до навчання створення ППЗ, оскільки компетентності, що будуть сформовані у студентів у межах даної дисципліни, матимуть значний вплив на процес навчання розроблення ППЗ. Шляхом аналізу навчального матеріалу можна визначити інваріантну частину, яка буде зустрічатися при вивченні ООП і при створенні ППЗ, це пов'язане з тим що, створення ППЗ тісно пов'язане з певними математичними об'єктами. Для вивільнення часу для більш докладного вивчення технологій і підходів до створення ППЗ виокремлену інваріантну частину бажано розглядати при навчанні ООП. Для пропедевтики створення ППЗ бажано при вивченні ООП розглядати задачі, які будуть базуватися на розгляді конкретних об'єктів певної предметної галузі, вдосконалення яких буде відбуватися в процесі вивчення окремих питань ООП.

Так студентам може бути запропоновано створити дерево класів, на вершині якого буде міститися клас "точка", від якого будуть створені два класи нащадки клас "коло" та клас "вектор". Клас "точка" має характеризуватися координатами в прямокутній декартовій системі координат (ПДСК), містити метод переміщення точки (задання нових координат точці) та метод друку характеристик точки. У класі "коло", координати точки, що є унаслідуваними з батьківського класу "точка", будуть розглядатися як координати центра кола, додатково клас "коло" має містити поле для задання радіуса та власний метод друку характеристик кола. Характеристиками кола, окрім координат центру та довжини радіуса, може виступати рівняння кола. Клас "вектор" буде розглядатися як радіус-вектор з початком в центрі ПДСК, тоді координати точки, які його характеризують, будуть розглядатися як координати кінця вектора. Характеристиками вектора, окрім координат його кінця, може виступати довжина вектора.

У процесі вивчення основних концепцій ООП і принципів використання об'єктно-орієнтованої технології програмування розглядувані класи можуть бути розширеними і уточненими, а послідовне вивчення та використання візуальних

компонент деякого середовища програмування надасть можливість поступового удосконалення інтерфейсу розроблюваного ПЗ.

Таке покрокове уточнення та розширення створюваних класів та побудованої програми на їх основі, дають уявлення про процес проектування та створення великих ПЗ. А використання математичних об’єктів та їх характеристик, на основі яких базувалися створені класи, сприяють кращому розумінню підходів до проектування дерев класів, що використовуються в реальних ППЗ.

Кінцеве реалізація створюваних класів та їх опис подано в додатку А.

2.4. Елементи навчально-методичного комплексу з дисципліни “Технології програмування та створення педагогічних програмних засобів”

Постає запитання, а як же ефективно вивчати технології створення ППЗ у педагогічному вищому навчальному закладі, щоб майбутні педагоги могли плідно брати участь у розробці власних ППЗ? Досвід показав, що таке вивчення повинно базуватися на прикладах вже розроблених реальних ППЗ [52]. В цьому випадку студенти ознайомляться з внутрішньою будовою таких проектів, ієрархією класів, особливостями реалізації методів опрацювання математичних чи фізичних даних. Проте необхідно розуміти, що немає потреби намагатися повністю відтворити розглядуваний ППЗ оскільки класи таких проектів мають велику кількість нюансів, важко реалізованих у межах навчального процесу. Тому дуже бажано, якщо це можливо, на основі реального ППЗ підготувати завдання, що пов’язані з навчальними ієрархіями класів, реалізаціями методів, які посильні для опанування студентами, що дозволить показати принципи і особливості розробки ППЗ.

У дослідженні запропоновано проводити таку підготовку на розгляді вже розроблених ППЗ і підготовлених на їх основі практичних та лабораторних завдань. Такими ППЗ взято ППЗ Numet, призначеного для підтримки навчання чисельних методів математики та ППЗ Gran2d – середовищ динамічної геометрії.

Зважаючи на вищесказане, нами було розроблено програму дисципліни “Технології програмування та створення педагогічних програмних засобів” в межах якої і має відбуватися підготовка майбутніх учителів до розроблення педагогічних програмних засобів. Вивчення даної дисципліни може проводитися

в межах варіативної (вибіркової) частини циклу “Професійна та практична (професійно-орієнтована) підготовка”.

Варто зазначити, що методична система навчання являє собою складну систему, яка залежить від багатьох чинників. Зокрема, зазначається обов’язковість певних компонент, а саме: цілі навчання, його зміст, методи, засоби та організаційні форми.

Цілі навчання пов’язані з набуттям певного рівня компетентностей. Структурними складовими таких компетентностей є відповідні знання, уміння та навички. Відповідно до яких і було визначено мету та завдання вивчення дисципліни.

Мета вивчення дисципліни полягає в ознайомленні студентів з методологією та технологіями створення ППЗ; формуванні уявлень про показники якості та вимоги до ППЗ; формуванні базових знань, які необхідні для проектування ППЗ; підготовці майбутніх учителів до самостійної роботи по оцінці та впровадженню в навчальний процес існуючих та власних ППЗ.

Завдання вивчення дисципліни: сформувати представлення про можливості використання ППЗ в майбутній професійній діяльності; ознайомити студентів з базовими поняттями проектування та розроблення ППЗ; ознайомити студентів з технологіями та особливостями створення ППЗ в межах певної предметної галузі; сформувати компетентності щодо розроблення та вдосконалення ППЗ; дати уявлення про експертизу якості ППЗ та сформувати первинні вміння проведення експертизи якості і педагогічної ефективності ППЗ.

В результаті засвоєння дисципліни *студент повинен*:

– знати загальні відомості про ППЗ; вимоги та показники якості, що встановлюються до ППЗ; особливості розробки ППЗ в межах певної предметної галузі; можливості використання ППЗ в навчальному процесі.

– мати уявлення про технології та парадигми програмування; класифікацію мов програмування; можливі інтегровні середовища розробки ППЗ.

– вміти проектувати та створювати ППЗ; якісно оцінювати розроблені ППЗ; добирати ППЗ відповідно до конкретної навчальної мети.

Зміст навчання. В цілому зміст навчання, як і інші його компоненти, визначається навчальною програмою. Тематика навчального матеріалу дисципліни “Технології програмування та створення ППЗ” містить 10 тем, які поділені на три змістовні модулі (таблиця 2.1).

Таблиця 2.1.

Структура дисципліни “Технології програмування та створення ППЗ”

Назви змістових модулів і тем	**Кількість годин**					
	усього	**у тому числі**				
		лек	**пр**	**лаб**	**інд**	**с.р.**
Змістовий модуль 1.						
Тема 1. Парадигми програмування: імперативна, декларативна, об’єктно-орієнтована.		1				2
Тема 2. Технології програмування: структурна, модульна, об’єктно-орієнтована.		1				2
Тема 3. Формалізація синтаксису й семантики мов програмування.						4
Разом за змістовим модулем 1	***12***	***2***				***10***
Змістовий модуль 2.						
Тема 4. Дидактичні основи створення і використання засобів інформаційно-комунікаційних технологій (ІКТ).		1				2
Тема 5. Класифікації ППЗ.		1				2
Тема 6. Вимоги до ППЗ, оцінка їх якості.						4
Тема 7. Особливості математичних ППЗ.		2				4
Разом за змістовим модулем 2	***16***	***4***				***12***
Змістовий модуль 3.						
Тема 8. Особливості ООП та його використання при створенні ППЗ, типові ієрархії класів, що можуть бути використані при створенні математичних ППЗ.		2	2			10
Тема 9. Особливості використання візуальних компонент для побудови інтерфейсу ППЗ.		2	2	2		5
Тема 10. Реалізація окремих класів, що можуть бути використані при створенні математичних ППЗ.		2	12	14		5
Разом за змістовим модулем 3	***62***	***6***	***16***	***16***		***24***
Усього годин	**90**	**12**	**16**	**16**		**46**

Наведемо більш розгорнутий погодинний розподіл практичних та лабораторних робіт які спираються на використання вихідних кодів ППЗ Numet та Gran2d (таблиця. 2.2.).

Таблиця 2.2.

Тематика практичних та лабораторних занять дисципліни “Технології програмування та створення ППЗ”

Тема заняття	Години		Примітка
	пр	лаб	
Особливості ООП та його використання при створенні ППЗ.	2		
Призначення та особливості інтерфейсу ППЗ Gran2d та Numet.	2		
Gran2d. Клас для графічного відображення прямокутної Декартової системи координат TGrPlace. / Numet. Ієрархія класів, загальні типи даних та константи. Клас TFunc.	2	2	Лабораторна робота №1
Gran2d. Ієрархія класів, загальні типи даних та константи. Базовий клас TParentObj. Клас “точка”. / Numet. Базовий клас TNumerMet. Клас TMetFuncs.	2	2	Лабораторна робота №2
Gran2d. Клас “лінія”. / Numet. Клас для задання функціональних залежностей у явному вигляді $y = f(x)$. Клас для задання звичайних диференціальних рівнянь у вигляді $y'(x) = f(x, y)$.	2	2	Лабораторна робота №3
Gran2d. Класи “коло” та “дуга”. / Numet. Клас для задання коефіцієнтів СЛАР.	2	2	Лабораторна робота №4
Gran2d. Класи текстового напису та підпису точки. / Numet. Клас для таблично заданої функції.	2	2	Лабораторна робота №5
Gran2d. Перерахунок залежних об’єктів. / Numet. Клас для генерування HTML документу при виведенні результатів обчислень.	2	4	Лабораторна робота №6
Підсумкове тестування		2	
Усього годин	16	16	

Бали, які студент може отримати, складаються з балів за розроблений ППЗ (60-70%), балів за тестування (20-30%) та решта балів за опрацювання завдань для самостійної роботи.

Процес навчання базується на розгляді вихідного коду одного з двох ППЗ (Gran2d чи Numet). Для виконання передбачено 6 лабораторних робіт, завдання яких є логічно та практично пов’язаними. Виконання запропонованих лабораторних робіт передбачає послідовне розроблення та вдосконалення окремих елементів ППЗ, тобто після виконуючи завданнь всіх лабораторних робіт студент має отримати працюючий ППЗ з визначеним набором послуг. Кожне завдання лабораторної роботи має 2 рівні (обовязковий та підвищений). Як було зазначено в пункті 2.2.1, виконання обов'язкового рівня надає можливість студенту отримати позитивну оцінку з дисципліни. Окрім зазначених лабораторних робіт студентам додатково пропонується розробити ще один ППЗ (орієнтовні завдання щодо розроблення додаткового ППЗ подано в додатку Б). Бали за ще один розроблений ППЗ виставлятися додатково.

Вивчення даної дисципліни дозволить здійснити фундаментальну підготовку майбутніх учителів математик та інформатики до самостійного розроблення ППЗ та використання ППЗ при організації і проведенні навчальних занять.

Засоби навчання. Під час підготовки майбутніх учителів математики та інформатики до розроблення ППЗ, як засоби навчання, в їх самому широкому розумінні, виступають мови та середовища програмування. Такими засобами навчання у межах методичної системи, що розглядається, є мова програмування FreePascal та відповідне середовище розробки Lazarus. Окрім того до засобів навчання відносяться і ППЗ Numet та Gran2d з їх вихідними кодами, на основі яких і відбувається відповідне навчання.

Методи навчання. Під час навчання у межах методичної системи, що розглядається, передбачається використання самого широкого діапазону навчальних методів. Досить детальний розгляд використовуваних методів навчання було подано в пункті 2.2.

Організаційні форми навчання. Досягнення цілей навчання обумовлюється не лише застосуванням тих чи інших методів, але й організаційними формами навчання.

Навчальний процес у межах методичної системи, що розглядається, реалізується в таких організаційних формах за дидактичною метою: навчальні заняття (лекції, практичні, лабораторні, консультації), виконання індивідуальних завдань (самостійна розробка додаткового ППЗ), самостійна робота (опрацювання питань, що виносяться на самостійний розгляд).

Якщо розглядати організаційні форми навчання за кількістю студентів то варто відмітити, що під час виконання лабораторних робіт, в різних групах, використовувалися підходи, як індивідуальної так і парної та колективної розробки ППЗ про які йшлося в пунктах 1.5.3 та 1.5.4.

З огляду на отримані в ході дослідження результати по формуванню компетентностей розроблення ППЗ групами з різною кількістю студентів, варто відмітити, що найбільш результативним було створення ППЗ в парі чи трійці. Така кількість студентів певною мірою обумовлена ходом розроблення ППЗ (завдання по створенню та вдосконаленню ППЗ пропонувалися послідовно і не були досить об’ємними). Присутність в колективі більше 3-х студентів не впливала на якість та швидкість виконання лабораторних завдань по розробці ППЗ, проте зменшувало якість отриманих компетентностей у студентів, оскільки не всі студенти мали змогу брати повноцінну участь у розробці. В порівнянні з індивідуальною розробкою ППЗ, розробка в парі чи невеликому колективі має ряд значних переваг: колективне обговорення та вирішення проблеми, швидше і ефективніше розв’язування задач, зменшення кількості помилок, більш ефективне внесення змін та ін.

2.4.1. Особливості математичних ППЗ. В процесі створення ППЗ, який буде використовуватися в межах деякої предметної галузі, розробник має розумітися на термінах, поняттях, правилах, методах опрацювання даних цієї предметної галузі.

Можна визначити кілька особливостей, які необхідно враховувати при створенні математичних ППЗ а саме:

- наявність відповідних математичних знань у розробника;
- можливість обчислення математичних та логічних виразів, заданих користувачем за допомогою ППЗ;
- можливість проведення символьних обчислень за допомогою ППЗ;
- врахування похибок при проведенні обчислень;
- виведення математичних виразів в “природному” вигляді;
- виконання графічних побудов.

Необхідною умовою створення математичних ППЗ є наявність у студента чи викладача, який буде створювати таке ППЗ, необхідної бази математичних знань з відповідної предметної галузі.

При використанні математичних ППЗ, у більшості випадків, користувачу необхідно працювати з математичними та логічними виразами, заданими власноруч. В зв’язку з чим постає задача в опрацюванні цих виразів. Такими виразами можуть бути:

- математичні вирази задані конкретними числовими значеннями (наприклад, 2+5*sin(30)) і необхідність їх обчислення;
- функціональні залежності однієї змінної (наприклад, sin(x)+cos(x), x*sin(x)+5) і необхідність обчислення їх значень при заданому аргументу;
- логічні вирази (наприклад, sin(3+5)>0, (sin(x)<cos(x)) or (cos(x)>0)) і необхідність визначення їх істинності чи хибності при заданих значеннях змінних.

Для опрацювання такого виразу необхідно провести синтаксичний аналіз текстового рядка, що містить вхідний вираз і перетворення текстових даних в спеціальну форму запису. Таке перетворення можна реалізувати декількома методами серед яких: представлення виразу у формі Бекуса-Наура, метод рекурсивного спуску та представлення виразу у формі польського запису (польській нотації) чи польського інверсного (зворотного) запису.

Більш детально зупинимося на використанні форми польського запису та форми польського інверсного запису, які на нашу думку, є легшим в розумінні та реалізації. Форма запису має назву «польський запис» в честь польського логіка Яна Лукасевича, яку він винайшов в 1920 році, щоб спростити логіку висловлювань. Польська інверсна форма запису була розроблена Чарльзом Хємбліном в середині 1950-х років на основі польської нотації. Особливістю даної форми запису є те що, символи операцій завжди йдуть відокремлено від своїх операндів (до або після) та відсутності дужок, а порядок виконання операцій визначається в залежності від розміщення операторів і є однозначним. Причому обчислення виразів, записаних в такій нотації, можна проводити шляхом одноразового перегляду з використанням стеку.

«Традиційна» форма запису математичних виразів, називається інфіксною формою запису, тобто являє собою запис математичного виразу, в якому знак операції знаходиться між його операндами (наприклад, (5 - 6) * 7).

Польський запис називається префіксною формою запису, тобто запис математичного виразу, в якому знак операції розміщений перед його операндами; (наприклад, * - 5 6 7).

Польський інверсний запис називається постфіксною формою запису, тобто запис математичного виразу, в якому операнди розміщені перед знаком їх операції; (наприклад, 5 6 - 7 *).

При формуванні польського чи інверсного польського запису в яких дужки не використовуються, важливо враховувати пріоритети арифметичних операцій:

- знак піднесення до степеня “^” – 4-й пріоритет;
- знаки множення та ділення “*”, “/” – 3-й пріоритет;
- знаки додавання та віднімання “+”, “-” – 2-й пріоритет.

1-й пріоритет досить часто віддається дужкам “(” та “)”, а 0-й пріоритет відносять до операндів.

У випадку, якщо вираз містить математичні функції (наприклад: ln, log, sin, cos, abs, ...), при побудові польського запису їх необхідно розглядати як оператори з максимальним пріоритетом. А в процесі виконання обчислення виразу, за даним

польським записом, функції необхідно розглядати як унарні операції з максимальним пріоритетом. Наприклад, вираз «sin(3+5*x)+cos(x)+log(tn(x+1)» буде записаний в інверсній польській формі, як «3 5 x * + sin x cos x 1 + tn log * +».

Для прикладу розглянемо алгоритми опрацювання польського інверсного запису які може використовувати розробник.

Побудова польського інверсного запису за інфіксною формою запису:

- Переглядаємо вхідний рядок зліва на право
  - Якщо зустрічається операнд, то копіюємо його в вихідний рядок
  - Якщо зустрічається відкриваюча дужка, то розміщуємо її в стек
  - Якщо зустрічаємо знак операції, то
    - Якщо стек порожнім, то розміщуємо знак операції в стек
    - Інакше
      - Якщо пріоритет останньої операції в стеці < пріоритету поточної операції, то розміщуємо знак операції в стек
      - Інакше
        - Записуємо знаки операції із стеку в вихідний рядок до тих пір, поки пріоритет останньої операції в стеці >= пріоритету поточної операції
        - Розміщуємо знак поточної операції в стек
  - Якщо зустрічається закриваюча дужка, то

    до тих пір поки верхнім елементом не стане відкриваюча дужка, виштовхуємо елементи з стеку в вихідний рядок. При цьому відкриваюча дужка знищується з стеку, але в вихідний рядок не додається дужка, що закривається
- Записуємо знаки операції з стеку в постфіксній запис до тих пір, поки стек не порожній.

Розглянемо застосування зазначеного алгоритму на конкретному прикладі. Нехай дано математичний вираз в інфіксній формі «a * (b + c) – d», який потрібно

перевести в постфіксну форму. Покрокове виконання переведення подано в таблиці 2.3.

Таблиця 2.3.

Переведення виразу з інфіксної в постфіксну форму

Елемент для аналізу	Виконувана операція	Результуючий рядок	Елементи стеку (вершина праворуч)		
a	Додаємо «a» до результуючого рядка	a			
*	Розміщуємо «*» в стек	a	*		
(	Розміщуємо «(» в стек	a	*	(	
b	Додаємо «b» до результуючого рядка	a b	*	(	
+	Розміщуємо «+» в стек	a b	*	(	+
c	Додаємо «c» до результуючого рядка	a b c	*	(	+
)	Виштовхуємо з стеку оператори до результуючого рядка поки не дійдемо до відкриваючої дужки. Дужку знищуємо з стеку	a b c +	*		
-	Оскільки пріоритет останньої операції в стеці «*» більший за пріоритет операції «-» то виштовхуємо операцію «*» з стеку в результуючий рядок. Розміщуємо «-» в стек	a b c + *	-		
d	Додаємо «d» до результуючого рядка	a b c + * d	-		
	Оскільки вхідний рядок порожній виштовхуємо операції з стеку в результуючий рядок	a b c + * d -			

В результаті отримаємо запис в постфіксній формі заданого виразу: «a b c + * d -».

Обчислення значення виразу, представленого в постфіксній формі.

- Переглядаємо постфіксний запис зліва на право
  - Якщо зустрічається операнд, то розміщуємо його в стек
  - Якщо зустрічається знак операції, то
    - Виконуємо цю операцію, використовуючи в якості операндів два числа з вершини стека

◦ Результат розміщуємо в стеці

• В вершині стеку знаходиться результат обчислення

Розглянемо застосування зазначеного алгоритму на конкретному прикладі. Обчислити значення виразу «2 3 5 + * 1 -» заданого в постфіксній формі. Покрокове виконання обчислення подано в таблиці 2.4

Таблиця 2.4.

Обчислення значення виразу заданого в постфіксній формі

Елемент для аналізу	Виконувана операція	Елементи стеку (вершина праворуч)		
2	Розміщуємо «2» в стек	2		
3	Розміщуємо «3» в стек	2	3	
5	Розміщуємо «5» в стек	2	3	5
+	Виконуємо операцію «+» над двома числами з вершини стеку «3» та «5» результат «8» розміщуємо в стек	2	8	
*	Виконуємо операцію «*» над двома числами з вершини стеку «2» та «8» результат «16» розміщуємо в стек	16		
1	Розміщуємо «1» в стек	16	1	
-	Виконуємо операцію «-» над двома числами з вершини стеку «16» та «1» результат «15» розміщуємо в стек	15		
	Оскільки вхідний рядок порожній результат обчислення в стеці	15		

В результаті отримаємо, як результат, число 15.

Перетворення значення виразу, представленого в постфіксній формі у інфіксну форму.

• Переглядаємо постфіксний запис зліва на право

▪ Якщо прочитано операнд, то заносимо його в стек

▪ Якщо прочитано знак операції, то

◦ Беремо два верхніх елемента з стеку

◦ Якщо в першому елементі пріоритет операції менше (і не рівний нулю) чим пріоритет розглядуваної операції, то беремо перший елемент в дужки

- Якщо в другому елементі пріоритет операції менше (і не рівний нулю) чим пріоритет розглядуваної операції, то беремо другий елемент в дужки
- Записуємо в стек рядок виду: 2-й елемент+знак операції+1-й елемент

• В вершині стеку знаходиться результат перетворення.

Розглянемо застосування зазначеного алгоритму на конкретному прикладі. Нехай дано математичний вираз в постфіксній формі «a b c + * d -», який потрібно перевести в інфіксну форму. Покрокове виконання переведення подано в таблиці 2.5.

Таблиця 2.5.

Переведення виразу з постфіксної в інфіксну форму

Елемент для аналізу	Виконувана операція	Елементи стеку (вершина праворуч)		
a	Розміщуємо «a» в стек	a		
b	Розміщуємо «b» в стек	a	b	
c	Розміщуємо «c» в стек	a	b	c
+	Беремо 2 верхні елементи стеку «c» та «b», оскільки вони є звичайними операндами (пріоритет рівний нулю) то в стек вносимо рядок «b+c»	a	b+c	
*	Беремо 2 верхні елементи стеку «b+c» та «a». «b+c» елемент з операцією «+», отже він має пріоритет операції «+» який менший за пріоритет операції «*» тому беремо його в дужки. «a» звичайний операнд (пріоритет рівний нулю). в стек вносимо рядок «a *(b+c)»	a*(b+c)		
d	«d» в стек	a*(b+c)	d	
-	Беремо 2 верхні елементи стеку «d» та «a*(b+c)». «d» звичайний операнд (пріоритет рівний нулю). «a*(b+c)» елемент з операцією «*» який більший за пріоритет операції «-». в стек вносимо рядок «a*(b+c)-d»	a*(b+c)-d		
	Оскільки вхідний рядок порожній результат обчислення в стеці	a*(b+c)-d		

В результаті отримаємо запис в інфіксній формі заданого виразу: «a * (b + c) – d».

Розглянуті вище алгоритми будуть стосуватися і логічних виразів. Пріоритети логічних операторів визначаються наступним чином:

- операції порівняння: (> ≥ = ≠ ≤ <) – 5-й пріоритет;
- логічне “ні” (not) – 4-й пріоритет;
- логічне “і” (and) – 3-й пріоритет;
- логічне “або” (or) – 2-й пріоритет.

В деяких випадках у користувача, при використання ППЗ, має бути можливість проведення символьних обчислень, реалізація алгоритмів яких є досить складною задачею для розробника, наприклад пошук похідної чи значення невизначеного інтегралу, а інколи і нерозв’язною. Такі обчислення також можуть базуватися на використанні польських форм записів.

В загалі кажучи, на даний час існує значна кількість вже готових реалізацій бібліотек та розширень для виконання обчислень математичних та логічних виразів і проведення символьних обчислень, що можуть використовуватися розробниками при створенні власних програмних засобів. Проте створення власних реалізацій зазначених обчислень є досить гарною практикою для розробника і в подальшому може надати можливість розширення та вдосконалення розроблених алгоритмів.

В процесі виконання математичних обчислень засобами мови програмування відбувається виникнення похибки таких обчислень. Реалізація певного алгоритму має власну похибку, яка включає в себе: похибки операторів; похибки методу, яка випливає з дискретного характеру будь-якого чисельного алгоритму; похибки округлення результатів, які пов’язані з використанням в обчислювальних машинах чисел з обмеженою точністю представлення. Якщо ж розглядати застосування ППЗ для аналізу певної моделі, то джерелами похибок, ще будуть похибки математичної моделі, які пов’язані з її невідповідністю предметній галузі та похибки вхідних даних, прийнятих для розрахунку.

В процесі функціонування ППЗ математичного призначення може виникнути необхідність у відображенні математичних виразів у “природному” вигляді. В залежності від формату документу, в який буде проводитися вивід, розв’язання даної задачі може бути різним. Так виведення може проводитися:

- в HTML документ, в який буде генеруватися вигляд необхідного математичного виразу використовуючи: таблиці, для задання різних рівнів формули; надрядкові та підрядкові символи, для вказання степенів та індексів; зміну шрифтів для відображення грецьких та спеціальних символів;
- в документ з TEX форматуванням чи форматування окремих формул використовуючи TEX розмітку для їх розміщення в HTML документі;
- в документ текстового процесора MS Word засобами microsoft equation чи документ офісного пакету LibreOffice використовуючи його роботу за рахунок COM технології.
- в документ графічного типу з збереженням в файлі і можливістю його подальшого використання.

Для виконання такого відображення математичних формул розроблена значна кількість бібліотек та розширень, наприклад: alTex, Ritex, ExprDraw, ExprMake.

При розробці ППЗ, в якому буде необхідно проводити графічні побудови (побудова графіків функцій, побудова геометричних об’єктів), необхідно пам’ятати про деякі особливості використання комп’ютера для виконання графічних побудов, а саме: відмінність системи координат графічної області виведення від звичної декартової системи координат та необхідність виведення результатів графічної побудови на пристрій з дискретною кількістю точок.

При роботі з графічними побудовами засобами комп’ютера доводиться працювати з двома системами координат. Перша система – це система координат графічної області виведення чи екранна система координат. Координатами точки в цій системі є номер пікселя в рядку X та номер рядка пікселів Y, де $0 \le X \le Xmax$, $0 \le Y \le Ymax$). Початок координат розміщується в лівому

верхньому кутку графічної області виведення. Параметри екранної системи координат (максимальне число пікселів в рядках *Xmax* та максимальне число рядків пікселів *Ymax*) залежать від поточного графічного режиму та розміру графічної області виведення. Таким чином, ця система координат певним чином пов'язана з конкретним графічним пристроєм та режимом його роботи.

Друга система координат – так звана декартова або математична. Вона представляє собою систему координат *(x, y)*, де $xmin \le x \le xmax$ та $ymin \le y \le ymax$, яку визначає програміст і яка є незалежною від конкретного графічного пристрою виведення.

Параметри, якими задаються діапазони зміни *x* і *y* (*xmin*, *ymin* , *xmax*, *ymax*), визначають прямокутну область в математичному двовимірному просторі. Ці параметри залежать лише від конкретної задачі.

Математичні координати і координати графічної області виведення зв'язані між собою простим відношенням:

$$X = X\max \cdot \left(\frac{x - x\min}{x\max - x\min} \right) \quad Y = Y\max \cdot \left[1 - \left(\frac{y - y\min}{y\max - y\min} \right) \right].$$

Формула для екранної координати *Y* дещо відрізняється від формули для координати *X* в силу того, що в екранній системі координат *OY* направлена вниз.

Що стосується другої особливості, то вона полягає в тому, що монітор, який буде відображувати побудову на деякій графічній області – це прилад з дискретною кількістю точок. Він має скінченну ширину та висоту і між будь-якими двома точками міститься скінченна і по математичним міркам, невелика кількість так званих елементів зображення – пікселів. Окрім того, координати всіх точок, які можна відобразити на екрані монітора, повинні мати цілочисельні значення. Природнім виходом з даної ситуації є округлення отриманої координати точки до найближчого цілочисельного значення. Зрозуміло, що виконуючи такі дії відбувається часткове спотворення зображення графіка функції чи графічного об'єкту.

2.4.2. Призначення та особливості інтерфейсу ППЗ Gran2d. В навчальній програмі з математики для загальноосвітніх навчальних закладів, 10-11 класи

(академічний рівень) [164] та навчальній програмі з математики для загальноосвітніх навчальних закладів, 10-11 класи (профільний рівень) [165] вказано, що підвищенню ефективності уроків з математики в старших класах сприяє використання програм Gran1, Gran2D, Gran3D, DG, EUREKA, бібліотек електронних наочностей тощо.

ППЗ Gran2d є складовою частиною програмного комплексу “Gran”, авторське право на який мають Жалдак М. І., Вітюк О. В. та Горошко Ю. В. [1]. Проте починаючи з 2006 року вдосконаленням характеристик програми Gran2d займається Костюченко А.О. під керівництвом Жалдака М. І. та Горошка Ю. В. Програма Gran2d є програмою динамічної геометрії і призначена для підтримки навчання планіметрії, як в школі так і в ВНЗ.

В програмі Gran2d передбачено можливості працювати з багатьма геометричними об’єктами: точка, лінія, коло, дуга, ламана. Кожен з зазначених об’єктів в свою чергу може використовуватися по різному. Так точка може бути одною з трьох типів: вільна точка – точка, координати якої задаються числовими значеннями, таку точку можна вільно переміщувати по робочій області; напівзалежна точка – точка, яка прив’язана до лінії чи кола, таку точку можна переміщувати лише вздовж того об’єкту, до якого її прив’язано; залежна точка – точка положення якої повністю визначається іншими об’єктами, які є базовими для неї. Залежною точка може бути, якщо вона є: точкою перетину деяких об’єктів, точкою дотику прямої до кола, точкою деякого геометричного перетворення (симетрії, повороту, гомотетії), точкою кінця дуги, аналітично заданою, точкою координати якої задані математичним виразом. Лінія, яку в подальшому будемо вважати прямою лінією, може виступати в ролі: прямої, відрізку, променя, бісектриси кута, лінія дотику до кола, перпендикулярна чи паралельна пряма. Коло може задаватися своїм центром та радіусом або центром та точкою на колі. Дуга може виступати ролі: власне дуги, сектора чи сегмента. Ламана у випадку коли є замкненою, може виступати в ролі многокутника.

Оскільки ППЗ Gran2d розроблявся для використання на уроках планіметрії, то в основі його функціонування покладена шкільна планіметрія. Основним

об’єктом є точка, яка виступає як базовий об’єкт для більшості інших планіметричних об’єктів, які в подальшому також можуть бути базовими об’єктами. Розглянемо на основі яких базових об’єктів можуть будуватися геометричні примітиви за рахунок ППЗ Gran2d:

- пряма, відрізок або промінь задається двома точками;
- перпендикулярна чи паралельна прямої задається точкою та деякою іншою лінією;
- бісектриса кута задається трьома точками, що визначають кут;
- дотична до кола задається точкою через яку мають проходити дотичні та колом;
- коло за точкою на колі задається точкою, що визначає центр кола та точкою, яка буде знаходитися на колі;
- коло за радіусом задається точкою, що визначає центр кола та двома точками, що визначатимуть початок та кінець радіуса;
- дуга, сектор, сегмент задаються п’ятьма точками (центр кола початок та кінець радіуса кола на якому відбувається побудова, точки, що визначають початок та кінець центрального кута)
- ламана задається послідовністю точок;
- середня точка задається двома точками;
- симетрична точка відносно точки задається точкою до якої буде будуватися симетрична та точкою, що визначатиме центр симетрії;
- симетрична точка відносно лінії задається точкою до якої буде будуватися симетрична та лінією, що визначатиме вісь симетрії;
- інверсна точка задається точкою до якої буде будуватися інверсна точка та колом
- точка перетину об’єктів задається двома об’єктами (лінія, коло, дуга).

За рахунок ППЗ Gran2d користувач також може:

- обчислювати відстань між двома точками, відстань між точкою та прямою, довжину кола чи ламаної, площу кола чи многокутника, градусну міру кута;

- виконувати геометричні перетворення створених геометричних об’єктів (поворот, гомотетію, паралельне перенесення, деформацію, симетрію відносно точки чи прямої);
- створювати та виконувати макроконструкції;
- будувати геометричне місце точок (ГМТ);
- проводити покрокове відображення послідовності виконання побудови.

Зважаючи на можливість переміщення вільних точок, які найчастіше виступають базовими для геометричних побудов, можна в реальному часі, змінюючи їх положення, досліджувати зміни отриманої побудови.

Досить вагоме значення для ППЗ має інтерфейс програми. ППЗ Gran2d базується на MDI-стандарті, коли в межах головного вікна програми існують декілька вікон для відображення відповідних даних. В головному вікні програми є меню для доступу до всіх послуг програми, панелі інструментів для пришвидшення доступу до основних послуг програми, а також 3 вікна (зображення, список об’єктів, динамічні вирази), розмір і положення яких можна змінювати в межах головного вікна (рис. 2.1).

Вікно “Зображення” призначене для розміщення (відображення) об’єктів програми – геометричних об’єктів, написів, обчислень, кнопок, тощо. Дане вікно містить: *поле підказки* – поле в верхній частині вікна, де з’являються короткі інструкції про те, яку дію необхідно виконати на поточному етапі роботи; *поле інформування* – це поле, де виводяться координати точки, що відповідає поточному положенню вказівника миші у вікні та основні дані про об’єкт над яким знаходиться вказівник миші; *область зображення (робоча область)* – область головного вікна програми, де зображуються створені об’єкти, розміщуються написи та кнопки. Кожен об’єкт, розміщений на робочій області, має власне контекстне меню, в якому відображаються послуги для роботи з відповідним об’єктом. Зважаючи на те, що ППЗ Gran2d є середовищем саме динамічної геометрії положення створених об’єктів може бути змінене користувачем методом переміщення відповідного об’єкту по робочій області в результаті положення залежних об’єктів та всі обрахунки автоматично

змінюються відносно нових характеристик об’єкту, що переміщується. Крім переміщення власне геометричних об’єктів по робочій області можуть бути переміщені і підписи точок, але в межах певного радіусу відносно самої точки.

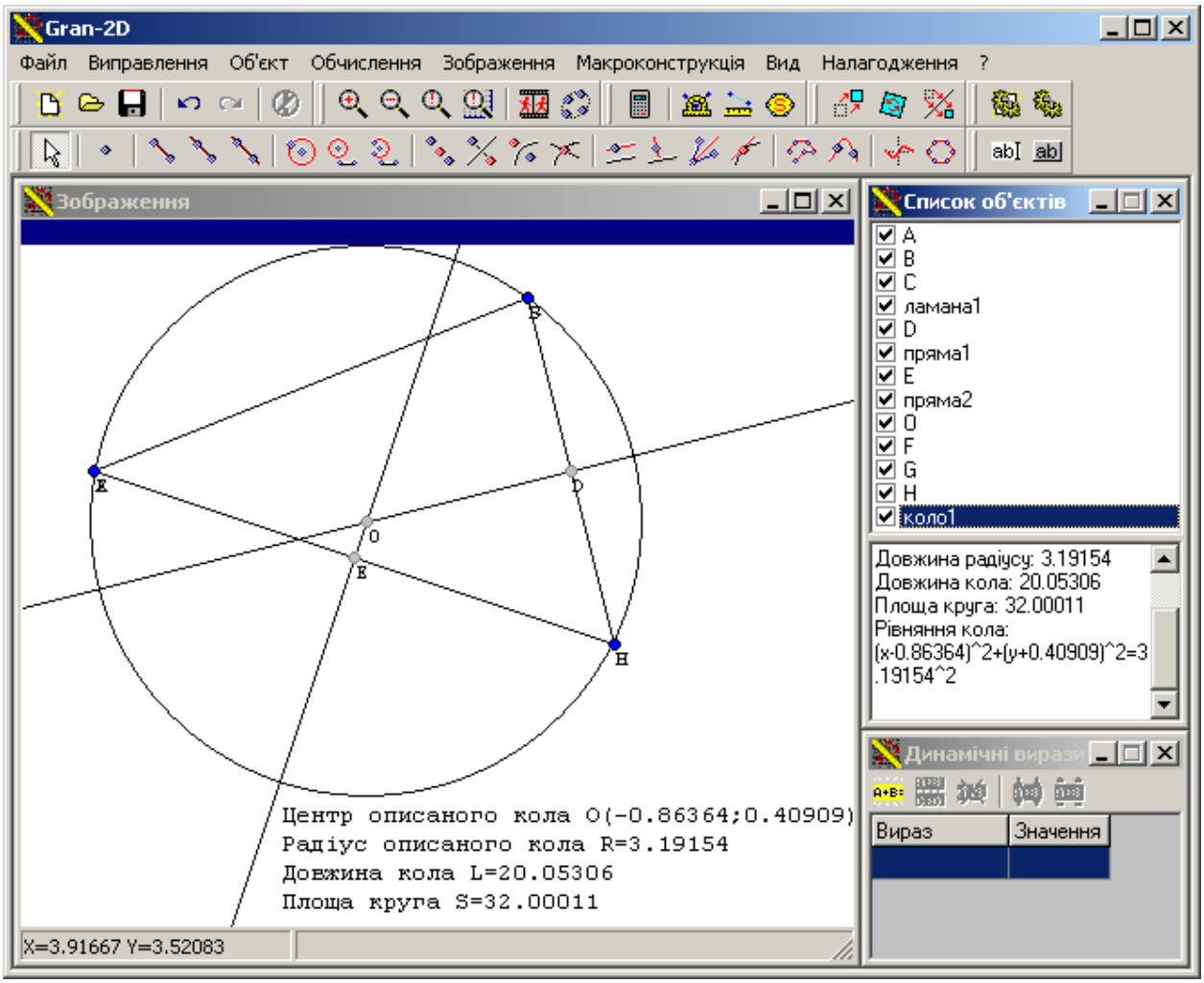


Рис. 2.1. Вікно ППЗ Gran2d

Крім того в робочій області можна переміщувати систему координат та змінювати масштаб, використовуючи мишу. Так для зміни масштабу достатньо покрутити колесо миші, а переміщення системи координат відбувається при переміщенні миші з затисненою правою кнопкою миші чи одночасним затисненням кнопки Ctrl та лівої кнопки миші. Робоча область може бути налагодження згідно вподобань користувача чи вимог певної задачі за рахунок використання послуги головного меню *“Налагодження/Параметри програми”*, відповідні налагодження зосереджені на закладці *“Графічна область”*. Для робочої області можна встановити:

- колір робочої області;

- зображення координатних осей (колір, тип лінії, відображення стрілок);
- зображення координатної сітки (колір, тип лінії);
- підписи осей;
- одиниці розбиття на осях (автоматичні або з фіксованим кроком, заданим в десятковому вигляді чи вигляді нескоротного дробу);
- виведення одиниць виміру на осях (не виводити, виводити лише одиничні, виводити всі);
- шрифт виведення підписів та одиниць виміру.

Вікно "Список об'єктів" містить перелік всіх геометричних об'єктів, що були створені або завантажені у процесі роботи з програмою. Біля кожного об'єкту розміщується перемикач ввімкнення якого відповідає необхідності відображати відповідний об'єкт в робочій області. Проте при необхідності бачити приховані об'єкти можна ввімкнути перемикач *"Зображення/Зображати сховані об'єкти"*.

У нижній частині вікна знаходяться область характеристик об'єкту, в якій виводяться деякі властивості поточного об'єкта (дані про базові об'єкти, числові характеристики об'єкту, алгебраїчні рівняння що визначають об'єкт тощо).

Вікно "Динамічних виразів" являє собою таблицю, що містить перелік назв створених динамічних виразів та їх обчислені значення. Динамічний вираз – це деякий математичний вираз, операндами якого можуть виступати числові характеристики геометричних об'єктів чи певні залежності між ними (довжини, площі, величини кутів т.д.). Враховуючи динамічність обчислень, що проводяться при необхідності окремі значення порахованих динамічних виразів можна зафіксувати скориставшись послугою *"Обчислення/Динамічний вираз/Зафіксувати поточне значення"*. Використання динамічних виразів в версії 2.0 програми частково забезпечується за рахунок використання динамічних вимірювань в написах, що розміщуються на робочій області (детальніше використання написів буде розглянуто пізніше).

Для управління положенням вікно призначена послуга головного меню *"Вид"*. Використовуючи перемикач *"Вид/Автоматичні розміри"* можна

розмістити підлеглі вікна без перекривання у головному вікні програми та автоматично змінювати їх розміри при зміні розмірів головного вікна програми. Якщо цей перемикач вимкнути, тоді можна довільно змінювати розміри та положення підлеглих вікон. Звернення до послуги "*Вид/Розміри положення за замовчуванням"* надає можливість повернутися до початкових характеристик вікон, запропонованих розробниками програми. Звернення до послуги "*Вид/Розміри за головним вікном"* надає можливість встановити розміри підлеглі вікна без перекривання у головному вікні програми та залишити можливість в подальшому змінювати положення та розміри підлеглих вікон. Окрім того дана послуга головного меню *"Вид"* містить перемикачі які можна використовувати для відображення чи приховування окремих панелей інструментів.

Для сучасних ППЗ досить важливою є робота з буфером обміну. В пункті головного меню "*Виправлення*" містяться послуги для роботи з буфером обміну. Послуга "*Виправлення/Скопіювати активне вікно*" призначена для копіювання зображення активного вікна (з заголовком) до буферу обміну. Послуга "*Виправлення/Скопіювати*" призначений для копіювання до буферу обміну даних, що залежать від того, яке вікно було поточним. Якщо поточним є вікно *"Зображення"* – до буферу обміну буде скопійовано графічне зображення з цього вікна (без рядка підказок та поля інформування). Якщо поточним вікном є вікно *"Список об'єктів"* – до буферу обміну буде скопійовано вираз, що відповідає поточному об'єкту. Якщо поточним вікном є вікно *"Динамічні вирази"* – до буферу обміну буде скопійовано дані (ім'я та значення) які відповідають поточному динамічному виразу.

Для встановлення загальних параметрів роботи з програмою використовується послуга головного меню "*Налагодження*". Дане меню містить такі послуги: "*Мова*" – для вибору мови інтерфейсу; *"Зберегти параметри"* – зберегти встановлені параметри програми до файлу налагодження і використовувати їх як параметри за замовчуванням; *"Параметри програми"* – призначена для відкриття допоміжного вікна налаштування параметрів програми

(рис. 2.2), яке містить: загальні налагодження, налагодження графічної області та налагодження оформлення окремих об’єктів.

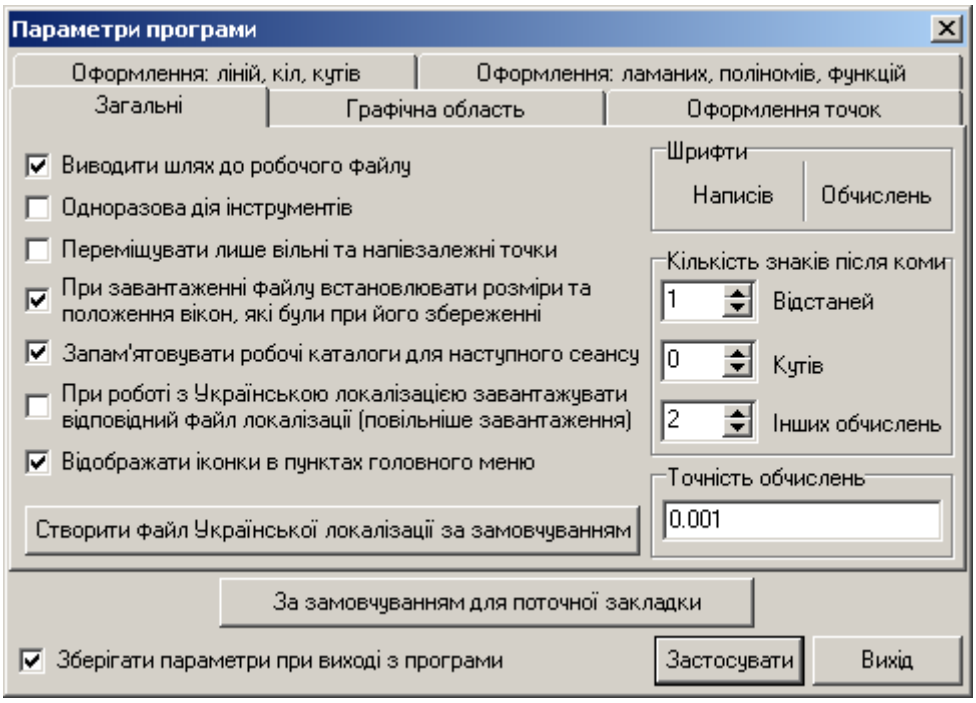


Рис. 2.2. Вікно налаштування параметрів ППЗ Gran2d

Створення нових геометричних об’єктів може відбуватися з екрану або через явне задання характеристик створюваного об’єкту за рахунок використання додаткового вікна з відповідними елементами інтерфейсу (загальний вигляд вікон для створення окремих геометричних об’єктів подано в додатку В).

При створенні об’єктів через явне задання характеристик необхідно скористатися відповідною послугою головного меню “*Об’єкт/Створення/…”* та в відкритому додатковому вікні задати характеристики створюваного об’єкту. Серед характеристик основне місце займає назва об’єкту та вибір базових об’єктів (кількість та типи цих об’єктів було розглянуто раніше). Окрім назви та базових об’єктів необхідно задати характеристики, які будуть визначати відображення створюваного об’єкту в робочій області. В залежності від типу створюваного об’єкту такими характеристиками є:

- для точки – колір лінії, необхідність та колір зафарбування точки, розмір точки (радіус зображуваної точки), колір та розмір підпису, необхідність відображати назву точки та її координати;

- для лінії – колір, товщина та тип лінії. У випадку створення відрізку можливість зображення стрілок на кінцях відрізку;
- для кола та ламаної – колір, товщина та тип лінії відображення, колір та тип зафарбування (прозоре, суцільне, діагональне і т.д.), спосіб зафарбування (звичайний чи напівпрозорий). Характеристики, що стосуються зафарбування до ламаної застосовуються лише тоді коли вона є замкненою.

Створення об’єктів через явне задання характеристик розглянемо на прикладі створення точки заданої аналітично, створення якої можливе лише через задання характеристик користувачем. Аналітично задана точка – це точка координати якої визначаються деякими математичними виразами. Для задання такої точки необхідно скористатися послугою *“Об’єкт/Створення/Аналітична точка”* і в відкритому додатковому вікні (рис. 2.3) вказати характеристики для нової точки (координати X, Y та характеристики, що визначатимуть відображення точки в робочій області). При аналітичному заданні точки в її координатах можуть бути використані математичні функції, а також функції вимірювання характеристик геометричних об’єктів значення яких автоматично обраховується. Серед функції вимірювання характеристик визначені за якими обчислюється: Len(A,B) – довжина відрізка між точками A і B; Len(A,Name) – відстань між точкою A і об’єктом (відрізок, промінь, пряма) з іменем Name; Len(Name) – довжина об’єкту (коло, дуга, ламана) з іменем Name; Angle(A, B, C) – міра кута в радіанах (від 0 до π), заданого точками A, B, C; OAngle(A,B,C) – міра орієнтованого кута в радіанах (від 0 до 2π), заданого точками A, B, C (міра кут вимірювання завжди проти годинникової стрілки); XAngle(A,B) – міра кута в радіанах нахилу вектора AB до осі OX (від 0 до 2π); X(A) – координата X точки A; Y(A) – координата Y точки A; Norm(A) – відстань від A до початку координат (полярний радіус точки); Arg(A) – кут, що утворено віссю абсцис і променем, який проведено з початку координат і проходить через точку A (полярний кут точки, від 0 до 2π); Area(A,B,C,…,N) – площа N-кутника (вершини перелічуються через кому; Area(Name) – площа об’єкту (коло, ламана, дуга) з іменем Name (у випадку коли

ламана незамкнена її площа дорівнює 0, якщо об’єктом є дуга то площа обчислюється залежно від типу дуги (сектор, сегмент, дуга).

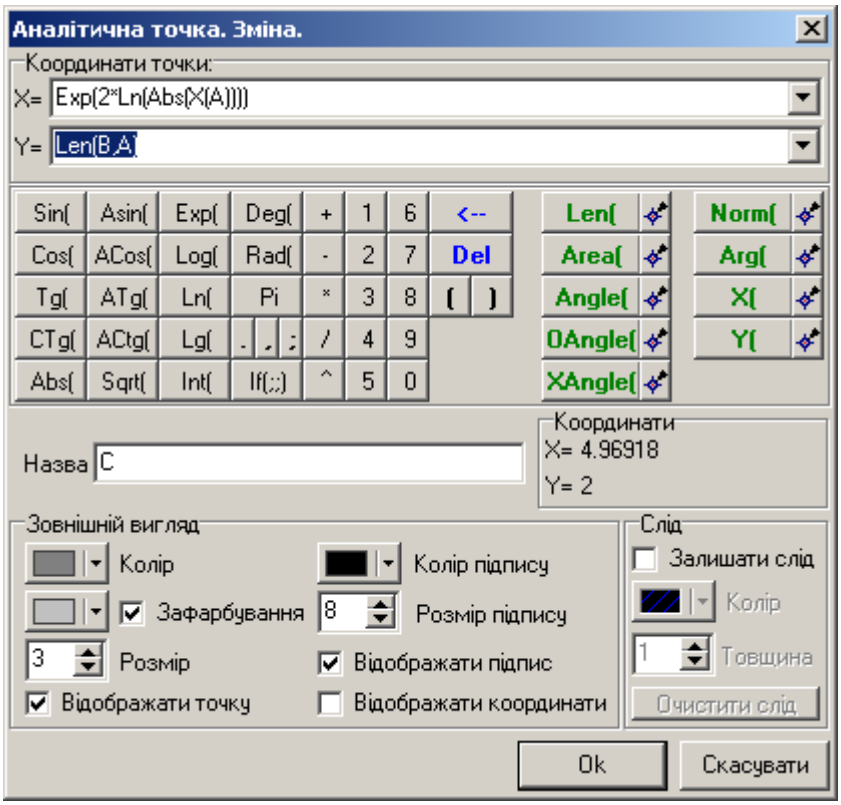


Рис. 2.3. Вікно встановлення характеристик аналітично заданої точки

Для створення нового об’єкту з екрану, необхідно обрати тип створюваного об’єкту на панелі інструментів чи обрати відповідну послугу головного меню *“Об’єкт/Створення з екрану/...”*. Після вибору типу створюваного об’єкту необхідно послідовно в області зображення вказати, натиснувши ліву кнопку миші, об’єкти на основі яких буде базуватися створюваний об’єкт. При створенні об’єктів в полі підказки, яке розміщене в верхній частині вінка *“Зображення”* буде виводитися повідомлення про виконувану операцію та необхідний базовий об’єкт в даний момент, окрім того курсор миші змінить свій вигляд при наведенні на об’єкт, який може бути використаний як базовий. У випадку, якщо відбувається створення точки чи є необхідність вибору точки як базового об’єкту то ліву кнопку миші можна натиснути на порожньому місці в якому має знаходитися дана точка, в результаті чого точка буде створена і використана як базова.

Для редагування створених об’єктів (тобто зміни їх характеристик) використовують послугу *“Змінити”* з контекстного меню вікна *“Список об’єктів”* чи контекстного меню відповідного об’єкту в області зображення. При цьому на екран викликається те саме допоміжне вікно, яке використовувалася при створенні об’єкту через явне задання характеристик.

Для вилучення об’єкту використовують послугу *“Вилучити”* з контекстного меню вікна *“Список об’єктів”* чи контекстного меню відповідного об’єкту в області зображення. У випадку, якщо об’єкт що вилучається виступає базовим для інших об’єктів то користувачу буде повідомлено про вилучення і залежних об’єктів, оскільки вони не зможуть існувати без об’єкту на основі якого їх створено.

У вікні *“Зображення”* крім розміщення геометричних об’єктів можуть бути розміщені написи та кнопки. Розміщенні написи та кнопки можна довільно переміщувати в межах вікна *“Зображення”* за допомогою мишки. Кожен розміщений напис та кнопка має власне контекстне меню за послугами якого відповідний напис чи кнопку можна змінити, вилучити, сховати чи закріпити. Закріплений напис чи кнопка прив’язуються до Декартових координат робочої області, таким чином при зміні масштабу чи переміщенні Декартової площини даний елемент буде змінювати своє положення, в свою чергу незакріплений напис чи кнопка має фіксоване розміщення в області зображення.

Для розміщення напису необхідно скористатися послугою *“Об’єкт/Додати напис”*. У вікні, що відповідає даній послузі (рис. 2.4) в відповідне поле необхідно ввести текст напису, для якого можу бути заданий шрифт та часткове його налагодження (гарнітура, розмір, стиль, колір). Крім загальних налагоджень форматування тексту, в написах можуть бути використані теги для форматування частин тексту, а саме:

- – напівжирний текст;
- – курсив;
- <u></u> – підкреслений;
- <s></s> – закреслений;

- – нижні індекси; – верхні індекси.

В написах також можуть бути використані специфічні математичні символи та літери грецького алфавіту. Окрім звичайного тексту в напис можуть бути додані вирази динамічних вимірювань, які заключаються в квадратних дужках «[», «]». Динамічне вимірювання – це вираз, який може містити числа, арифметичні операції, елементарні функції, а також функції обчислення характеристик геометричних об'єктів. Значення автоматично обрахованого динамічного вимірювання додається до напису. В написах крім функцій обчислення характеристик об'єктів, що були розглянуті при розгляді створення точки, заданої аналітично, може бути використана функція Name(A) за якою повертається назва точки A. За рахунок використання даної функції можлива автоматична зміна назв точко в написі при перейменуванні самої точки.

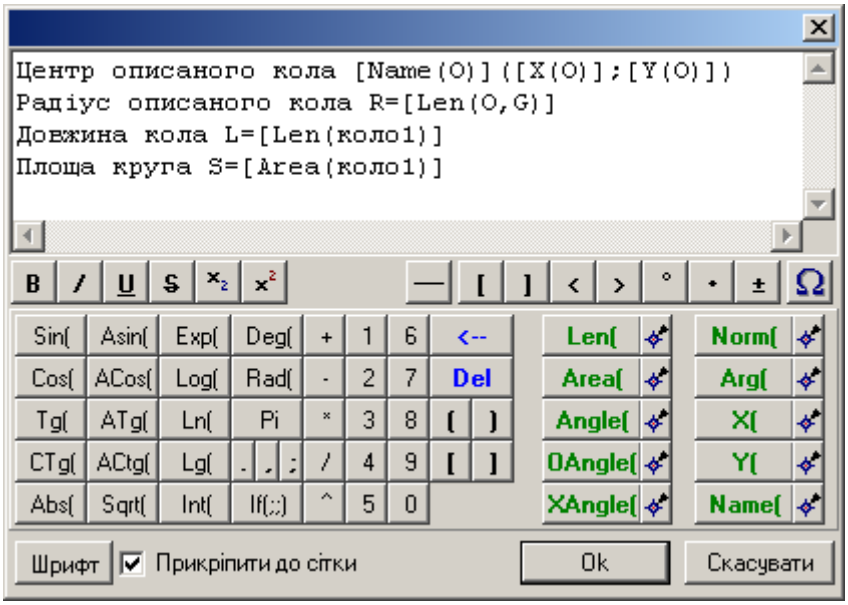


Рис. 2.4. Вікно встановлення характеристик напису

Для розміщення кнопки необхідно скористатися послугою *"Об'єкт/Додати кнопку"*. У вікні, що відповідає даній послузі (рис. 2.5) необхідно задати підпис кнопки, вибрати її тип та вказати відповідні характеристик згідно обраного типу. Кнопка може бути таких типів:

- показати/сховати об'єкти – натиснення на дану кнопку будуть послідовно відображувати та приховувати вказані об'єкти розміщені в вікні

“Зображення”. Для налагодження кнопки даного типу необхідно вибрати бажані об’єкти;

- показати повідомлення – натиснення на дану кнопку виводить текстове повідомлення, яке вказується в відповідному полі;
- відкрити файл – натиснення на дану кнопку відкриває файл, вказаний в відповідному полі.

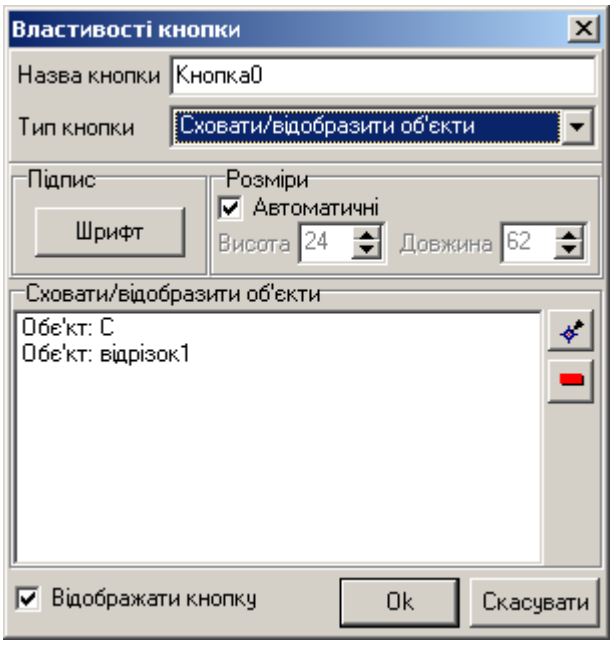
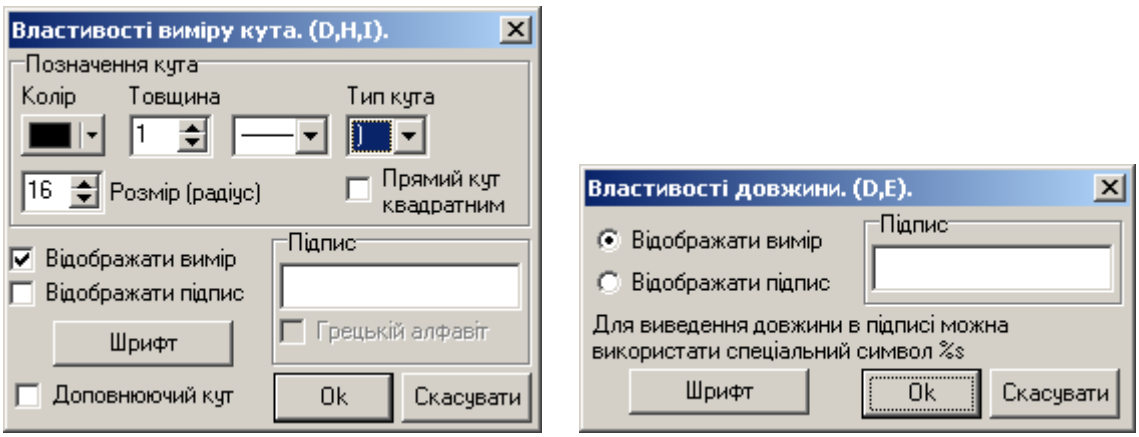


Рис. 2.5. Вікно встановлення характеристик кнопки

Над створеними геометричними об’єктами у користувача є можливість виконувати певні обчислення та проводити параметричні перетворення. Послуги для виконання обчислень згруповані в пункті головного меню *“Обчислення”*. За рахунок відповідних послуг можна обчислювати:

- градусну міру кута чи довжину відрізка в результаті чого створюється відповідний об’єкт, що розміщуються на робочій області, який має власні характеристики для налагодження відображення (рис. 2.6);
- довжини чи площі кола та ламаної в результаті чого створюється напис, що розміщується на робочій області, в який автоматично генерується необхідний динамічний вираз.
- значення математичного виразу, значення визначеного інтегралу на заданому відрізку чи значення похідної функції в точці. Такі обчислення проводяться в межах додаткового вікна і не призводять ні до яких змін.



а) б)

Рис. 2.6 Вікна налагодження відображення вимірів а) кута; б) довжини відрізку

Для виконання параметричного перетворення необхідно скористатися послугою *“Об’єкт/Перетворення параметрично”*. За рахунок відповідних послуг можна виконувати такі перетворення, як: поворот, паралельне перенесення, гомотетію, деформацію, симетрію відносно точки чи прямої. У вікні, що відповідає даній послузі (рис. 2.7) перш за все необхідно вибрати вхідних об’єкт до якого буде застосоване перетворення та встановити характеристики для обраного перетворення.

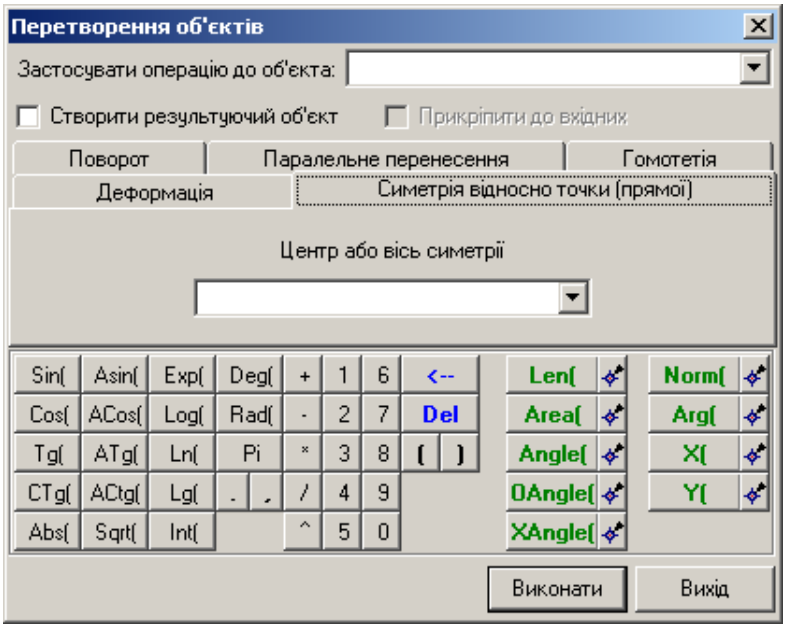


Рис. 2.7. Вікно встановлення характеристик для перетворення об’єкту

Окрім для виконання перетворення можна встановити додаткові налагодження. Зокрема якщо перемикач *“Створити результуючий об’єкт”* не включено то після перетворення вхідний об’єкт буде переміщено згідно

вибраного перетворення, у випадку ж ввімкнення даного перемикача вхідний об’єкт залишиться незмінним проте буде створено новий результуючий об’єкт, який буде відповідним перетворенням вхідного об’єкту. Також при ввімкненні перемикача *“Створити результуючий об’єкт”* активізується перемикач *“Прикріпити до вхідних”* ввімкнення якого призведе до зв’язування точок вхідного та вихідного об’єктів, в результаті чого при подальшій зміні положення базових точок вхідного об’єкту буде відбуватися зміна положення відповідних точок і вихідного об’єкту згідно проведеного перетворення.

Для огляду характеристик ППЗ Gran2d розглянемо приклад: “Побудувати описане коло навколо трикутника заданого координатами вершин A(-4; 1) B(1; 3) C(2, -1) та визначити характеристики отриманого кола (довжина радіуса, довжина кола та площа кругу)”.

Для побудови кола описаного навколо трикутника, необхідно знайти точку, що буде визначати центр цього кола, який знаходиться на перетині серединних перпендикулярів.

Трикутник, що заданий координатами своїх вершин побудуємо за рахунок побудови замкненої ламаної, для цього скориставшись послугою *“Об’єкт/Створення/Ламана”* і в відкритому додатковому вікні задамо послідовно координати точок, що будуть визначати вершини даного трикутника, вкажемо, що створювана ламана має бути замкненою та встановимо інші характеристики для ламаної згідно рисунку 2.8.

Побудуємо серединні перпендикуляри до утворених сторін трикутника BC та CA. Для цього скористаємося інструментом  *“Створення середньої точки”* і вказавши послідовно точки між якими потрібно подувати середню точку, отримаємо точки D і E – середини сторін BC та CA, відповідно. Після цього використавши інструмент  *“Створення перпендикуляру до прямої”* і вказавши точку D а потім сторону BC отримаємо перпендикуляр, який і буде серединним перпендикуляром до сторони BC. Аналогічно будуємо серединний перпендикуляр до сторони CA, що проходить через точку E.

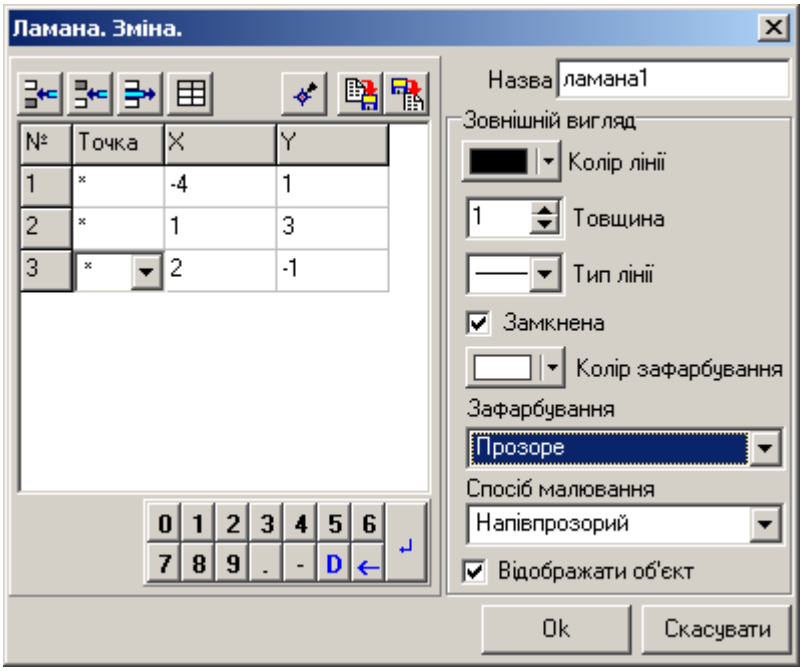


Рис. 2.8. Вікно встановлення характеристик ламаної

Побудуємо точку, яка відповідатиме центру описаного кола, як точку перетину побудованих перпендикулярів. Для цього скориставшись інструментом *“Створення точки перетину об’єктів”* та вказавши послідовно утворені серединні перпендикуляри отримаємо точку. Відкривши вікно характеристик утвореної точки змінимо її назву на «О».

Побудуємо описане коло. Для цього скориставшись інструментом *“Створення кола”* та вказавши точку О, як центр кола і будь яку вершину трикутника, як точку на колі, отримаємо коло описане навколо трикутника.

Для виведення характеристик кола на екран створимо відповідний напис з динамічними вимірюваннями. Для цього скориставшись інструментом *“Додати напис”* в відкрите додаткове вікно створення напису введемо текст згідно рисунку 2.4.

В результаті отримаємо побудову зображену на рисунку 2.1.

2.4.3. Методика навчання розробки ППЗ з геометрії на прикладі ППЗ Gran2d. Для формування у студентів компетентностей з розробки ППЗ розгляд особливостей програмного коду будемо проводити у дещо спрощеному вигляді. Проте це надає можливість побудувати каркас програми, який в подальшому можна буде доповнити новими характеристиками. Реалізації окремих методів

класів, що розглядаються та деяких підпрограм загального призначення, винесені до додатку Д.

Клас для графічного відображення прямокутної декартової системи координат TGrPlace.

Зважаючи на специфіку ППЗ даного виду, основною необхідністю є візуалізація геометричних побудов, отже необхідно визначити клас, який буде містити характеристики робочої області – прямокутної декартової системи координат (ПДСК). Таким класом є `TGrPlace`, який буде базуватися на класі `TImage`.

```
type TGrPlace=class(TImage)
  Private
    . . . { опис полів та методів згідно властивостей }
    Fxc, Fyc: Integer;
    FRolX, FRolY: Integer;
    FZoom: Extended;
    FPixelsSm: Integer;
    FlagDrawScene: Boolean;
    LastX, LastY: Double;
    AB: Array[1..5] Of Record
       ex, ey: Extended;
     End;
  Public
    constructor Create(AOwner: TComponent); override;
    procedure ScreenToAbsZ(x, y: Integer; var xx, yy: Extended);
    procedure AbsToScreenZ(x, y: Extended; var xx,yy: Extended);
        overload;
    procedure AbsToScreenZ(x, y: Extended; var xx, yy:Integer);
        overload;
    procedure SetZoom(n: Extended);
    procedure DrawLine(x1,y1,x2,y2:Extended; Lt: Integer=0);
    procedure Draw;
    procedure GrMouseMove(Sender: TObject; Shift: TShiftState; X,
        Y: Integer);
```

```
    procedure GrMouseDown(Sender: TObject; Button: TMouseButton;
        Shift: TShiftState; X, Y: Integer);
    procedure GrMouseUp(Sender: TObject; Button: TMouseButton;
        Shift: TShiftState; X, Y: Integer);
    procedure Save(f: TIniFile; Sec: String);
    procedure Load(f: TIniFile; Sec: String);
    property ColorObl: TColor read FColorObl write FColorObl;
    property PutOsi  : Boolean read FPutOsi write FPutOsi;
    property ColorOsi: TColor read FColorOsi write FColorOsi;
    property DrawGrid: Boolean read FDrawGrid write  FDrawGrid;
    property XNameOsi: String read FXNameOsi write FXNameOsi;
    property YNameOsi: String read FYNameOsi write FYNameOsi;
    property TecDX: Extended read FTecDX;
    property TecDY: Extended read FTecDY;
    property PixelsSm: Integer read FPixelsSm;
End;
```

Для налагодження візуального відображення області в цей клас включені наступні властивості:

`ColorObl: TColor` – колір фону робочої області;

`PutOsi: Boolean` – необхідність відображення осей координат;

`ColorOsi: TColor` – колір осей координат;

`DrawGrid: Boolean` – необхідність відображення координатної сітки;

`XNameOsi: String` та `YNameOsi: String` – підписи осей;

`FlagDrawScene: Boolean` – необхідність перемальовувати графічну область;

`LastX, LastY: Double` – в даних полях зберігаються координати попереднього положення курсору миші при виконанні переміщення системи координат за допомогою миші.

Деякі з властивостей є властивостями лише для читання (в їх описі відсутня частина `write`), оскільки їх значення не можуть змінюватися напряму за межами даного класу. Так наприклад за рахунок `PixelsSm: Integer` – визначається кількості точок графічної області в 1 см., яка залежить від параметрів екрану; за

рахунок властивостей `TecDX: Extended` та `TecDY: Extended` – визначаються координати точки в ПДСК, над якою знаходиться курсор миші.

Оскільки при виконанні побудов є необхідність працювати як з координатами в ПДСК так і з екранними координатами, у класі визначено методи для переведення координат з ПДСК в координати робочої області та навпаки `ScreenToAbsZ`, `AbsToScreenZ`. Дані методи використовують деякі поля з секції класу private: `Fxc, Fyc: Integer` – середина графічної області; `FRolX, FRolY: Integer` – кількість пікселів на яку зсунуто початок координат ПДСК відносно центру графічної області; `FPixelsSm, FZoom: Extended` – коефіцієнт в скільки разів необхідно змінити кількість точок в 1 см. (масштаб).

Метод ScreenToAbsZ перерахунку екранних координат x, y в координати ПДСК, він може бути використаний для реалізації відображення математичних координати при русі мишки над графічної областю.

Метод AbsToScreenZ переведення координат x, y заданих в ПДСК в екранні координати, він може бути використаний для реалізації виконання побудов геометричних об'єктів.

При роботі з графічною областю, може виникнути необхідність змінювати масштаб, для цього призначено метод `procedure SetZoom(n: Extended)`, який надає можливість змінювати коефіцієнт кількості точок в 1 см. в n разів.

Даний клас містить поле-масив записів, в якому будуть міститися координати кутів робочої області в ПДСК.

```
AB: Array[1..5] Of Record
  ex,ey: Extended;
End;
```

Даний масив містить 5 елементів, незважаючи на 4 кути робочої області. П'ятий кут буде співпадати з першим, це спеціально зроблено для спрощення в подальшому певних обчислень.

Значення даного поля будуть перераховуватися в залежності від ширини (`Width`) та висоти (`Height`) графічної області, наступним чином:

```
with AB[1] do ScreenToAbsZ(0,0, ex,ey);
with AB[2] do ScreenToAbsZ(Width,0, ex,ey);
```

```
with AB[3] do ScreenToAbsZ(Width,Height, ex,ey);
with AB[4] do ScreenToAbsZ(0,Height, ex,ey);
AB[5]:=AB[1];
```

Зважаючи на те, що встановлені користувачем параметри графічної області можуть знадобитися йому при наступному використанні ППЗ, передбачено методи для збереження і завантаження цих параметрів в текстовий файли спеціального виду – так звані ini-файли:

```
procedure Save(f: TIniFile; Sec: String);
procedure Load(f: TIniFile; Sec: String).
```

Для виконання побудови лінії за її типом (пряма – Lt=0, промінь – Lt=1, відрізок – Lt=2) до класу графічної області включено метод:

```
procedure DrawLine(x1,y1,x2,y2:Extended; Lt: Integer=0);
```

Під час побудови лінії виникає необхідність пошуку точок перетину даної лінії з лініями, що визначають межі робочої області. Для пошуку точки перетину двох ліній кожна з яких задається координатами двох точок передбачено підпрограму `X_Line_Line(x11,y11, x12,y12, x21,y21, x22,y22: Extended; var x,y: Extended): Boolean`. У випадку, якщо лінії не перетинаються, тобто є паралельні, дана підпрограма повертає `False`.

Одним з основних методів класу `TGrPlace` є метод `Draw`, який призначений для перемалювання робочої області, а саме відображення координатної сітки, осей координат та їх підписів.

Серед методів, визначених в даному класі, також присутні методи, які відносяться до опрацювання певних подій, зокрема подій, що пов’язані з використанням миші:

`procedure GrMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer)` – метод опрацювання події переміщення курсору миші над робочою областю, за рахунок даного методу реалізована можливість зміни положення системи координат в межах робочої області

`procedure GrMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)` – метод опрацювання події натиснення кнопок миші над робочою областю.

`procedure GrMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)` – метод опрацювання події відпускання кнопок миші над робочою областю.

Дані методи використовуються для відображення поточних координат ПДСК чи характеристик об'єкту під вказівником миші, виділення об'єкту в робочій області, переміщення об'єкту, виклику відповідного контекстного меню та ін.

Після розгляду класу, який визначає об'єкт графічної області студентам пропонується виконати наступне завдання.

Завдання 1 (ППЗ Gran2d).

Обов'язковий рівень. Розробити клас для відображення ПДСК (`TgrPlace`), розмістивши його в модулі `UgraphObj`.

```
TGrPlace=class(TImage)
 Private
  . . . { опис полів та методів згідно властивостей }
  Fxc, Fyc: Integer;
  FRolX, FRolY: Integer;
  FZoom: Extended;
  FPixelsSm: Integer;
  FTecDX, FTecDY: Extended;
  FlagDrawScene: Boolean;
  LastX, LastY: Double;
  AB: Array[1..5] Of Record
     ex, ey: Extended;
   End;
 Public
  constructor Create(AOwner: TComponent); override;
  procedure ScreenToAbsZ(x, y: Integer; var xx, yy: Extended);
  procedure AbsToScreenZ(x, y: Extended; var xx, yy:Extended);
    overload;
  procedure AbsToScreenZ(x, y: Extended; var xx, yy:Integer);
    overload;
  procedure SetZoom(n: Extended);
  procedure DrawLine(x1,y1,x2,y2:Extended; Lt: Byte=0);
  procedure Draw;
```

```
    procedure GrMouseMove(Sender: TObject; Shift: TShiftState; X, Y:
      Integer);
    property ColorObl: TColor read FColorObl write FColorObl;
    property PutOsi  : Boolean read FPutOsi write FPutOsi;
    property ColorOsi: TColor read FColorOsi write FColorOsi;
    property TecDX: Extended read FTecDX;
    property TecDY: Extended read FTecDY;
    property PixelsSm: Integer read FPixelsSm;
  End;
```

Створити програму з графічним інтерфейсом для відображення ПДСК, описаної в класі `TgrPlace`. Додати до програми головне меню; створити послугу головного меню для проведення налагодження ПДСК. Область ПДСК має автоматично перемальовуватися при зміні розмірів вікна. В конструкторі програми передбачити автоматичне створення і розміщення екземпляру описаного класу, в деструкторі програми передбачити знищення створеного екземпляру класу.

Зауваження. Описаний клас `TgrPlace` можна оформити в вигляді окремої компоненти та додати її до списку компонент Lazarus і в подальшому використовувати компоненту, а не окремий клас.

Підвищений рівень. Виконати завдання обов'язкового рівня та доповнити клас `TgrPlace` методом `GrMouseMove` – для реалізації відображення поточних координат графічної області під вказівником миші та можливості переміщення (зміни положення) системи координат в межах робочої області; методами `Save` та `Load` – для збереження і завантаження параметрів графічної області за рахунок використання текстових файлів спеціального виду (ini-файлів).

Ієрархія класів, загальні типи даних та константи. Перемалювання графічної області й створених об'єктів.

Розглянемо тепер, як здійснюється опис і реалізація основних об'єктів програми Gran2d, таких як точка, лінія, дуга, коло, ламана. Сукупність цих об'єктів утворюють ієрархію, на вершині якої знаходиться клас `TParentObj` в якому зосередженні загальні характеристики розглядуваних об'єктів (рис. 2.9). В

подальшому специфічні характеристики окремих геометричних об’єктів будуть визначатися в класах нащадках.

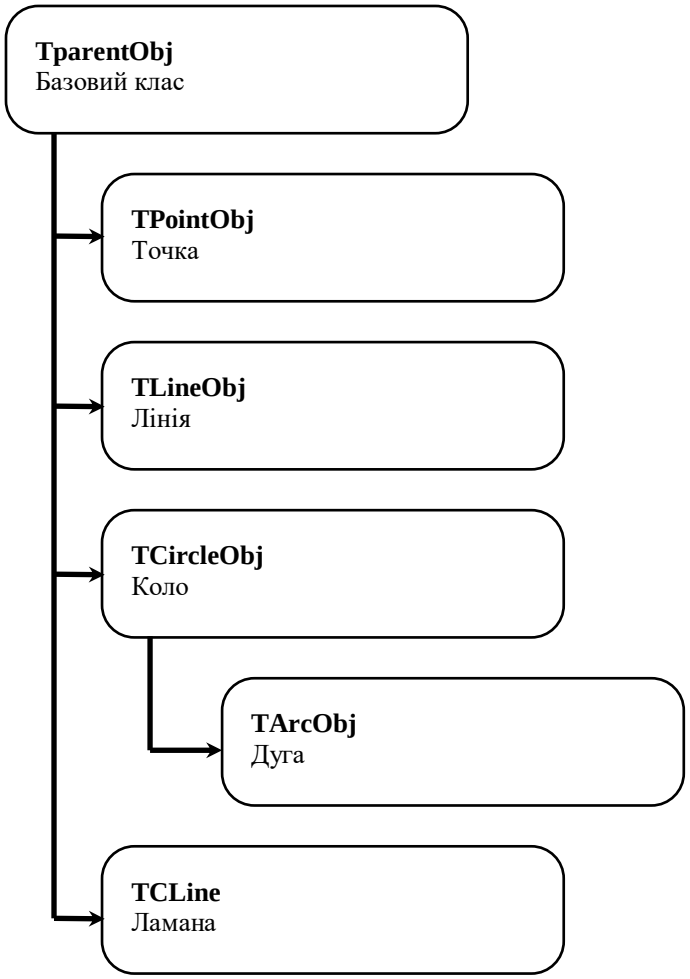


Рис. 2.9. Дерево класів геометричних об’єктів програми Gran2d

Перш ніж переходити до опису класів геометричних об’єктів, визначимо необхідні типи даних та константи.

```
Type
  ShapeType=(shNone, shDot, shLineVidriz, shLinePromin, shLine,
        shLinePar, shLinePerp, shCircle,shCircleR, ...)
```

Перелічуваний тип `ShapeType` містить перелік назв типів для можливих геометричних об’єктів, які використовуються у ППЗ Gran2d.

```
Const
  PStyles : Array[0..4] of TPenStyle=(psSolid, psDash, psDot,
        psDashDot, psDashDotDot);
  BStyles : Array[0..7] of TBrushStyle=(bsClear, bsSolid,
        bsHorizontal, bsVertical,bsFDiagonal, bsBDiagonal, bsCross,
        bsDiagCross);
```

Масиви PStyles та BStkyles визначають перелік стилів ліній та стилів зафарбування, які можуть використовуватися під час відображення геометричних об’єктів на графічній області.

Для розміщення створених об’єктів в середовищі ППЗ Gran2d використовується об’єкт (клас) ListObj: TCheckListBox, в кожному елементі списку якого буде міститися програмний об’єкт, що відповідає деякому геометричному об’єкту. Наявність відмітки біля елемента списку відповідає необхідності відображувати відповідний геометричний об’єкт на графічній області.

Для перемалювання графічної області та всіх створених об’єктів передбачено процедуру ReDraw за якою спочатку перемальовується координатна площина, а потім всі об’єкти.

```
procedure ReDraw;
Var i: Integer;
begin
  GrPlace.Draw;
  For i:=0 To FMain.ListObj.Items.Count-1 Do
   (FMain.ListObj.Items.Objects[i] as TparentObj).Draw;
end;
```

Базовий клас TParentObj. Клас “точка”.

При створенні ППЗ Gran2d було визначено клас TParentObj, який буде батьківським для всіх інші класів геометричних об’єктів.

Опис даного класу має наступний вигляд:

```
TParentObj=class(TObject)
 private
  . . . { опис полів та методів згідно властивостей }
 protected
  function GetTypeName: String; virtual; abstract;
 public
  RefCount: Integer;
  Ref: Array[1..5] Of Integer;
  GrPlace: TGrPlace;
  Constructor Create;
```

```
    procedure ShowSett(M: TStrings; Move: Boolean=False); virtual;
        abstract;
    function ShowInfo: String; virtual; abstract;
    procedure Draw; virtual; abstract;
    procedure Save(f: TIniFile; Sec: String); virtual;
    procedure Load(f: TIniFile; Sec: String; List: TStrings);
        virtual;
    property TypeObj: ShapeType read FTypeObj write FTypeObj;
    property Name: String  read FName write FName;
    property Visible: Boolean  read FVisible write FVisible;
    property Select: Boolean read FSelect write FSelect;
    property ReCalc: Boolean read FReCalc write FReCalc;
  End;
```

В даному класі визначено такі властивості: будь-який об’єкт повинен мати ім’я `Name: String`, належати до певного типу `TypeObj: ShapeType`, містити покажчики чи є об’єкт в даний момент видимим в межах графічної області `Visible: Boolean` та виділеним `Selected: Boolean`. Об’єкт може залежати від інших об’єктів, наприклад, лінія залежить від двох точок, в такому випадку ці точки є базовими для лінії. Для цього введено поле `RefCount: Integer` для визначення кількості базових об’єктів та поле `Ref: Array[1..5] Of Integer`, яке буде відповідати за зв’язок об’єкту з його базовими об’єктами. При використанні об’єкту ламана, розмір цього масиву може бути збільшено.

Зрозуміло, що при зміні положення точки, яка є базовою, наприклад, для прямої, дана пряма також має змінити власне положення. Проте дана зміна має відбутися лише після перерахунку координат точки. Оскільки такий перерахунок може використовуватися з різних частин програми, введено поле `ReCalc: Boolean`, яке буде містити покажчик того, що характеристики об’єкту перераховані і можна приступати до перерахунку об’єктів, залежних від даного.

Зважаючи на те, що ППЗ базується на графічному відображенні об’єктів, даний клас містить поле, що є посиланням на об’єкт графічної області `GrPlace: TGrPlace`, визначений раніше.

При визначенні методів для класу `TParentObj` було передбачено надати можливість користувачу роботи з файлами, збереження та завантаження об’єктів. Для цього призначені методи, які в нащадках будуть довизначатися в залежності від введення нових характеристик нащадка:

```
procedure Save(f: TIniFile; Sec: String); virtual;
procedure Load(f: TIniFile; Sec: String; List: TStrings); virtual.
```

Крім того, передбачені методи, за якими будуть виводити короткі `ShowInfo: String` та розширені `ShowSett(M: TStrings; Move: Boolean=False)` дані, що характеризують об’єкт. Короткі характеристики можуть виводитися під час наведення курсору миші на об’єкт, розширені – при виділенні об’єкту в списку об’єктів і передаватися параметром `M: TStrings`. Метод `Draw` призначений для відображення об’єкту на графічній області.

Більшість з зазначених методів є віртуальними, це необхідно для можливості їх подальшої підміни в класах-нащадках. Крім того, деякі з методів є абстрактними, оскільки їх не можна визначити для даного класу, що не відповідає жодному реальному геометричному об’єкту.

Серед методів є захищена функція

```
function GetTypeName: String; virtual; abstract;
```

за якою повертається текстова назва типу об’єкту. Дана функція використовується при запису об’єкту у файл.

Після того, як буде описаний батьківський клас з визначеними загальними властивостями, можна приступати до опису похідних класів, тобто класів-нащадків, що будуть описувати певний геометричний об’єкт, або знову бути деяким проміжним класом для визначення і доповнення деяких особливостей групи інших геометричних об’єктів.

Так, наприклад, при визначенні класу “точка” базовий клас повинен бути доповнений новими властивостями, серед яких можна визначити:

- координати точки в ПДСК – `DPx`, `DPy` та на графічній області – `DGx`, `DGy` (оскільки буде необхідність використовувати як математичні координати

точки для виконання певних обчислень, так і екранні координати для зображення точки на графічній області;

- властивості за якими буде відбуватися відображення точки на робочій області, такі як розмір зображення точки (радіус) – `Dsize: Integer`, колір лінії, яка буде зображати точку – `Dcolor: TColor`, колір зафарбування точки – `DFill: TColor` та необхідність цього зафарбування – `DfillShow: Boolean`;
- поле, яке буде відповідати за підпис (назву) точки та його відображення `DLab: TTextPoint` (визначення класу `TTextPoint` буде показано пізніше). На початковому етапі підпис точки може виступати звичайною текстовою змінною і виводитися в межах графічної області відповідними методами;
- параметри підпису: необхідність відображення підпису взагалі – `DnameShow: Boolean` та необхідність дописування координат – `DcoorShow: Boolean`.

Також необхідно підмінити (довизначити) методи, описані в батьківському класі та при необхідності додати нові.

Крім того, в класі “точка” передбачено перезавантажений конструктор для можливості створення об’єкту “точка” з відповідними координатами.

Опис класу, який буде відповідати за геометричний об’єкт “точка”, має наступний вигляд:

```
TPointObj=class(TParentObj)
private
  . . . { опис полів та методів згідно властивостей }
  function GetTypeName: String; override;
  procedure ShowCaption;
public
  DPx, DPy  : Integer;
  DLab: TTextPoint;
  constructor Create(TypeDot: ShapeType; APlace: TGrPlace=nil);
      overload;
  constructor Create(x1, y1: Double; TypeDot: ShapeType; APlace:
      TGrPlace=nil); overload;
```

```
    Destructor Destroy; override;
    function ShowInfo: String; override;
    procedure ShowSett(M: TStrings; Move: Boolean=False); override;
    procedure Draw; override;
    procedure Save(f: TIniFile; Sec: String); override;
    procedure Load(f: TIniFile; Sec: String; List: TStrings);
        override;
    property DGx: Double  read FDGx write SetDGx;
    property DGy: Double  read FDGy write SetDGy;
    property DColor: TColor read FDColor write FDColor;
    property DSize: Integer read FDSize write FDSize;
    property DFill: TColor read FDFill write FDFill;
    property DFillShow : Boolean read FDFillShow write FDFillShow;
    property DNameShow: Boolean read FDNameShow write FDNameShow;
    property DCoorShow: Boolean read FDCoorShow write FDCoorShow;
  End;
```

Розглянемо призначення методів даного класу.

Метод `procedure ShowCaption` призначений для формування та відображення підпису точки в залежності від встановлених налагоджень: вивід лише назви точки чи з додавання координат розміщення.

Методи:

```
procedure SetDGx(const Value: Double);
procedure SetDGy(const Value: Double);
```

будуть викликатися при зміні координат точок в ПДСК і перераховувати ці координати в відповідні координати графічної області, для подальшого відображення точки та перераховувати позицію розміщення підпису.

Для відображення точки на графічній області в класі `TPointObj` підмінено метод малювання геометричного об’єкту `Draw`.

Крім методу `Draw`, необхідно визначити методи виведення даних про об’єкт `ShowInfo` та `ShowSett`, метод повернення назви типу об’єкту `GetTypeName` та довизначити методи збереження та завантаження об’єкту в відповідності до його нових характеристик.

Після розгляду даних класів студентам можна запропонувати наступне завдання.

Завдання 2 (ППЗ Gran2d).

Обов'язковий рівень. Розробити клас `TParentObj`, розмістивши його в модулі `UParentObj`:

```
TParentObj=class(TObject)
private
  . . . { опис полів та методів згідно властивостей }
public
  RefCount: Integer;
  Ref: Array[1..2] Of Integer;
  GrPlace: TGrPlace;
  Constructor Create;
  procedure ShowSett(M: TStrings; Move: Boolean=False); virtual;
    abstract;
  procedure Draw; virtual; abstract;
  property TypeObj: ShapeType read FTypeObj write FTypeObj;
  property Name: String  read FName write FName;
  property Visible: Boolean  read FVisible write FVisible;
End;
```

Розробити клас "точка" (`TPointObj`), розмістивши його в модулі `UPointObj`:

```
TPointObj=class(TParentObj)
private
  . . . { опис полів та методів згідно властивостей }
public
  DPx, DPy  : Integer;
  DLab: String;
  constructor Create(TypeDot: ShapeType; APlace: TGrPlace=nil);
    overload;
  constructor Create(x1, y1: Double; TypeDot: ShapeType; APlace:
    TGrPlace=nil); overload;
  Destructor Destroy; override;
  procedure ShowSett(M: TStrings; Move: Boolean=False); override;
  procedure Draw; override;
```

```
    property DGx: Double  read FDGx write SetDGx;
    property DGy: Double  read FDGy write SetDGy;
    property DColor: TColor read FDColor write FDColor;
    property DFill: TColor read FDFill write FDFill;
  End;
```

Доповнити програму, створену в попередньому завданні, додавши до неї: компоненту `TListBox` – для зберігання в ньому списку створюваних об'єктів; компоненту `TMemo` – для відображення даних про поточний об'єкт в списку об'єктів; послуги головного меню для створення деякої точки (створення нової точки має відбуватися через допоміжне вікно з вибором можливих налагоджень для об'єкту "точка"). Створені точки, як окремі об'єкти, мають розміщуватися в списку об'єктів. Описати підпрограму, яка б могла використовуватися для перемальовування області побудови та всіх створених об'єктів, описати виклик цієї підпрограми в необхідних місцях коду для правильного функціонування програми (чи замінити нею виклик перемальювання області ПДСК). При відображенні точки, поряд з нею має відображатися і її назва. Передбачити виведення даних про поточний об'єкт в списку об'єктів в компоненту `TMemo`.

Підвищений рівень. Виконати завдання обов'язкового рівня та доповнити класи `TParentObj` та `TPointObj` методами `Save` та `Load` – для збереження і завантаження властивостей створюваних об'єктів за рахунок використання текстових файлів спеціального виду (ini-файлів). Доповнити програму перевіркою, яка б унеможливлювала створення декількох точок з однаковими іменами. Довизначити клас `TParentObj` віртуальним, абстрактним методом для виведення коротких даних, які будуть характеризувати об'єкт: `ShowInfo: String`. Перевизначити даний метод в класі `TPointObj`. Доповнити клас `TGrPlace` та програму послугою аналізу знаходження курсора миші над об'єктом "точка" в межах графічної області, тобто коли курсор миші знаходиться над об'єктом в графічній області, необхідно відображати в рядку статусу коротку інформацію про об'єкт.

Клас “лінія”.

Далі необхідно розглянути зі студентами клас, який відповідатиме геометричному об’єкту “лінія”. Опис даного класу матиме вигляд:

```
TLineObj=class(TParentObj)
private
  Ax, By, Cz: Extended;
  FName1: String;
  FName2: String;
  . . . { опис полів та методів згідно властивостей }
  function GetTypeName: String; override;
public
  Gx1, Gy1, Gx2, Gy2: Extended;
  Px1, Py1, Px2, Py2: Integer;
  constructor Create(TypeLine: ShapeType; APlace: TGrPlace=nil);
  function ShowInfo: String; override;
  procedure ShowSett(M: TStrings; Move: Boolean=False); override;
  procedure SetPropLine2Dot(xx1, yy1, xx2, yy2: Extended);
  procedure SetPropLineDotLine(xx1,yy1, xx11, yy11, xx22, yy22 :
      Extended);
  procedure Draw; override;
  procedure SetNameDot(s1,s2: String);
  procedure Save(f: TIniFile; Sec: String); override;
  procedure Load(f: TIniFile; Sec: String; List: TStrings);
      override;
  property LStyle: Integer read FLStyle write FLStyle;
  property LColor: TColor read FLColor write FLColor;
  property LSize: Integer read FLSize write FLSize;
End;
```

Серед специфічних властивостей, даний клас містить:

- коефіцієнти `Ax`, `By`, `Cz` рівняння заданої прямої;
- параметри налагодження зображення прямої: стиль (`LStyle`), колір (`LColor`), товщина (`LSize`);
- координати точок, на яких базується пряма в ПДСК `Gx1`, `Gy1`, `Gx2`, `Gy2` та відповідні координати цих точок в графічній області `Px1`, `Py1`, `Px2`, `Py2`;

- `FName1`, `FName2` – назви точок, на яких базується лінія, для формуванні даних про характеристики об'єкту, метод `SetNameDot` для надання значень цим змінним, оскільки поля є захищеними.

Крім того клас `TLineObj` містить метод `SetPropLine2Dot`, за рахунок якого визначаються коефіцієнти рівняння лінії за двома точками, які її задають.

При створенні лінії не на основі двох точок, а наприклад перпендикулярно чи паралельно до деякої іншої лінії введено метод `SetPropLineDotLine`, який перераховує коефіцієнти рівняння лінії на основі базової точки та базової лінії.

Для класу "лінія", як і для класу "точка" необхідно визначити та довизначити методи, оголошені в батьківському класі.

Так необхідно перевизначити метод відображення лінії `Draw` для відображення саме лінії та метод `ShowSett` за рахунок якого відбувається формування в тектовому вигляді розширених характеристик про відповідну лінію.

Для закріплення знань для студентів необхідно запропонувати наступне завдання.

Завдання 3 (ППЗ Gran2d).

Обов'язковий рівень. Розробити клас "лінія" (`TLineObj`), розмістивши його в модулі `ULineObj`:

```
TLineObj=class(TParentObj)
private
  . . . { опис полів та методів згідно властивостей }
  Ax, By, Cz: Extended;
  FName1: String;
  FName2: String;
public
  Gx1, Gy1, Gx2, Gy2: Extended;
  Px1, Py1, Px2, Py2: Integer;
  constructor Create(TypeLine: ShapeType; APlace: TGrPlace=nil);
  procedure ShowSett(M: TStrings; Move: Boolean=False); override;
  procedure SetPropLine2Dot(xx1, yy1, xx2, yy2: Extended);
  procedure Draw; override;
  procedure SetNameDot(s1,s2: String);
```

```
    property LColor: TColor read FLColor write FLColor;
    property LSize: Byte read FLSize write FLSize;
  End;
```

Доповнити програму, створену в попередньому завданні, передбачивши можливість створення прямої чи відрізку, використовуючи додаткове вікно з можливістю вибору базових точок серед вже створених точок як програмних об’єктів та встановленням характеристик для створюваного об’єкту. Створені об’єкти мають розміщуватися в списку об’єктів.

Підвищений рівень. Виконати завдання обов’язкового рівня та доповнити клас `TLineObj`: можливістю створення променя; методами `Save` та `Load` – для збереження і завантаження властивостей створюваних об’єктів за рахунок використання текстових файлів спеціального виду (ini-файлів). Перевизначити метод `ShowInfo` в класі `TLineObj`. Доповнити клас `TGrPlace` та програму послугою аналізу знаходження курсора миші над об’єктом “пряма” в межах графічної області, тобто коли курсор миші знаходиться над об’єктом в графічній області, необхідно відображати в рядку статусу коротку інформацію про об’єкт.

Класи “коло” та “дуга”.

Наступним розглянутим класом буде клас, який відповідатиме графічному об’єкту коло (`TCircleObj`). Опис даного класу матиме вигляд:

```
  TCircleObj=class(TParentObj)
  private
    . . . { опис полів та методів згідно властивостей }
    function GetTypeName: String; override;
  public
    Gx1, Gy1, Gx2, Gy2, Gx3, Gy3 : Extended;
    Px1, Py1, Px2, Py2, Px3, Py3 : Integer;
    constructor Create(TypeLine: ShapeType; APlace: TGrPlace=nil);
    function ShowInfo: String; override;
    procedure ShowSett(M: TStrings; Move: Boolean=False); override;
    procedure SetPropCircle(xx1, yy1, xx2, yy2, xx3, yy3 :
        Extended);
    procedure Draw; override;
    procedure Save(f: TIniFile; Sec: String); override;
```

```
    procedure Load(f: TIniFile; Sec: String; List: TStrings);
        override;
    property CFill: TColor read FCFill write FCFill;
    property LStyle: Integer read FLStyle write FLStyle;
    property CStyle: Integer read FCStyle write FCStyle;
    property CMethFill: Integer read FCMethFill write FCMethFill;
    property LenR: Extended read FLenR;
    property LenLen: Extended read FLenLen;
    property LenArea: Extended read FLenArea;
    property Rivn: String read FRivn;
End;
```

Серед специфічних властивостей, даний клас містить:

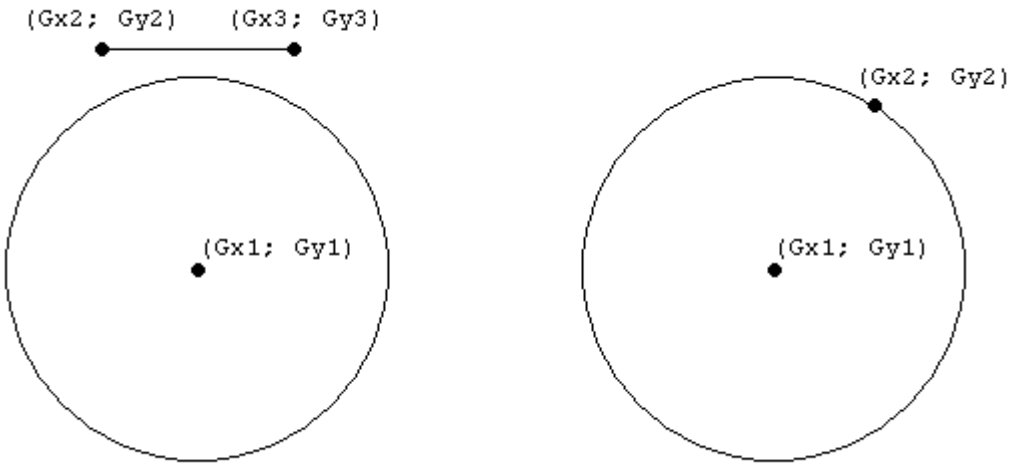
- параметри побудови зображення кола: `LStyle` – стиль лінії кола (суцільна, пунктирна, точками і т.д.), `CStyle` – стиль зафарбування кола (суцільне, діагональне і т.д.), `CFill` – колір зафарбування, `CMethFill` – стиль зафарбування (якщо містить 1, то зафарбування відбувається напівпрозоро);
- координати точок, на яких базується коло в ПДСК `Gx1, Gy1, Gx2, Gy2, Gx3, Gy3` – координати центра кола та координати точок, що визначатимуть радіус кола або координати точки на колі (рис. 2.10) та відповідні координати робочої області `Px1, Py1, Px2, Py2, Px3, Py3`.

Окрім цього, клас `TCircleObj` містить поля, за якими можна визначати характеристики створеного геометричного об’єкту: `LenR: Extended` – довжина радіуса кола, `LenLen: Extended` – довжина кола, `LenArea: Extended` – площа круга обмеженого колом, `Rivn: String` – рівняння кола. Ці характеристики будуть визначатися в межах методу `SetPropCircle`.

Даний клас також містить підмінені методи батьківського класу, за якими визначається його функціональність.

Клас “дуга” (`TArcObj`) в порівнянні з класом “коло” розширюється лише властивостями `Gx4, Gy4, Gx5, Gy5, Px4, Py4, Px5, Py5` що задають точки,

початку та кінця дуги (рис. 2.11). Також в класі “дуга” будуть підмінені відповідні батьківські методи.



а) коло задане радіусом б) коло задане точкою на колі

Рис. 2.10. Призначення змінних при заданні кола

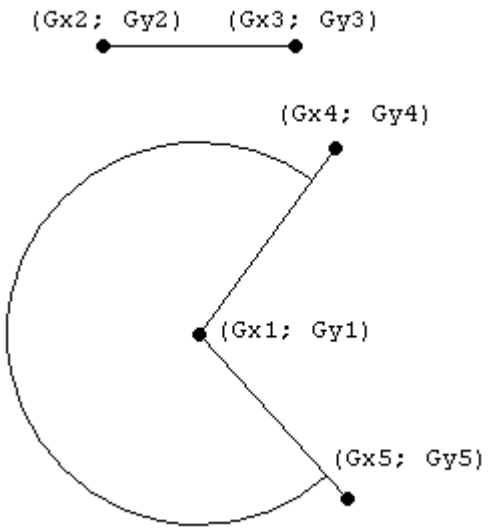


Рис. 2.11. Призначення змінних при заданні дуги

Студентам можна запропонувати наступне завдання.

Завдання 4 (ППЗ Gran2d).

Обов’язковий рівень. Розробити клас “коло” `TCircleObj`, розмістивши його в модулі `UCircObj`.

```
TCircleObj=class(TParentObj)
private
  . . . { опис полів та методів згідно властивостей }
```

```
    function GetTypeName: String; override;
  public
    Gx1, Gy1, Gx2, Gy2, Gx3, Gy3 : Extended;
    Px1, Py1, Px2, Py2, Px3, Py3 : Integer;
    constructor Create(TypeLine: ShapeType; APlace: TGrPlace=nil);
    procedure ShowSett(M: TStrings; Move: Boolean=False); override;
    procedure SetPropCircle(xx1, yy1, xx2, yy2, xx3, yy3 :
        Extended);
    procedure Draw; override;
    property CFill: TColor read FCFill write FCFill;
    property LStyle: Integer read FLStyle write FLStyle;
    property CStyle: Integer read FCStyle write FCStyle;
    property LenR: Extended read FLenR;
    property LenLen: Extended read FLenLen;
    property LenArea: Extended read FLenArea;
    property Rivn: String read FRivn;
  End;
```

Доповнити програму створену в попередньому завданні, передбачити можливість створення кола заданого центром та радіусом, використовуючи додаткове вікно з можливістю вибору базових точок серед вже створених точок як програмних об’єктів та встановленням характеристик для створюваного об’єкту. Створені об’єкти мають розміщуватися в списку об’єктів.

Підвищений рівень. Виконати завдання обов’язкового рівня та доповнити клас `TCircleObj` методами `Save` та `Load` – для збереження і завантаження властивостей створюваних об’єктів за рахунок використання текстових файлів спеціального виду (ini-файлів). Розробити клас “дуга” (`TArcObj`) та доповнити програму можливістю створення об’єкту класу “дуга”. Перевизначити метод `ShowInfo` в класах `TCircleObj` та `TArcObj`. Доповнити клас `TGrPlace` та програму послугою аналізу знаходження курсора миші над об’єктами “коло” та “дуга” в межах графічної області, тобто коли курсор миші знаходиться над об’єктом в графічній області, необхідно відображати в рядку статусу коротку інформацію про об’єкт.

Класи текстового напису та підпису точки

Крім дерева класів для геометричних об’єктів з класом `TParentObj` у вершині, в ППЗ Gran2d визначене дерево класів для текстових написів, які можуть бути розміщені в графічній області, з класом у вершині `TTextObj`. Серед таких текстових написів в програмі можуть зустрічаються: підпис точки, підписи обчислень кутів та відстані між точками, текстовий напис, який може використовуватися як пояснення до побудови. Дерево класів текстових написів подано на рисунку 2.12.

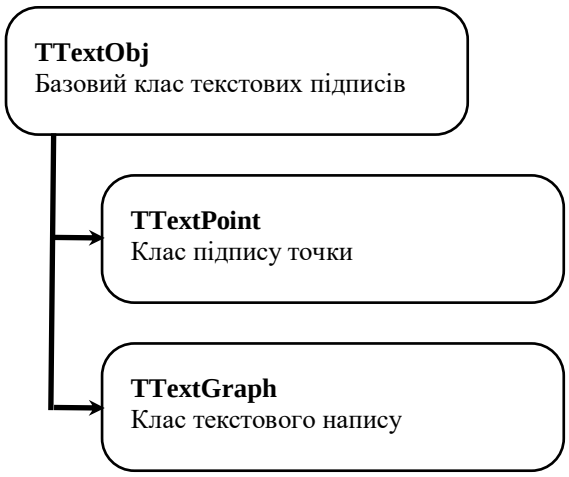


Рис. 2.12. Дерево класів текстових об’єктів програми Gran2d

Опис класу `TTextObj` має вигляд:

```
TTextObj=class(TLabel)
private
  XOld, YOld: Integer;
  FlagMove: Boolean;
  procedure LabelMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);virtual;
  procedure LabelMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);virtual;
  procedure LabelMouseMove(Sender: TObject; Shift: TShiftState;
      X,Y: Integer); virtual;
public
  GrPlace: TGrPlace;
  constructor Create(AOwner: TComponent); overload; override;
  constructor Create(AOwner: TComponent; APlace: TGrPlace);
      overload;
```

```
    procedure SetPopMenu(APopupMenu: TPopupMenu);
    procedure Save(f: TIniFile; Sec: String); virtual; abstract;
    procedure Load(f: TIniFile; Sec: String; List: TStrings);
        virtual; abstract;
  end;
```

Цей клас, як і його нащадки містить посилання на графічну область `GrPalce: TGrPlace`.

Серед методів класу `TTextObj` визначено метод `SetPopMenu`, який призначений для підключення власного контекстного меню до напису. Для послуги збереження та завантаження даних про написи визначено абстрактні методі `Load` та `Save`, які будуть підмінені в нащадках. Зважаючи на необхідність переміщувати текстові написи по графічній області, в даному класі визначено методи опрацювання подій миші `LabelMouseDown`, `LabelMouseMove`, `LabelMouseUp`, які використовують поля `XOld, YOld: Integer` для зберігання координат попереднього положення напису та `FlagMove: Boolean` покажчика, який буде мати значення `True` в момент переміщення напису.

Нащадками від `TTextObj` є класи для підпису точки `TTextPoint` та для текстового напису для пояснень `TTextGraph`.

Опис класу для підпису точки має наступний вигляд:

```
  TTextPoint=class(TTextObj)
  public
     Dx, Dy: Integer;
     FWith, FHeight: Integer;
     procedure GraphCaption(const Value: String; AColor: TColor;
      ASize: Integer; AName: String);
     procedure LabelMouseMove(Sender: TObject; Shift: TShiftState;
      X, Y: Integer);override;
  end;
```

Опис класу текстового напису для пояснень має наступний вигляд:

```
  TTextGraph=class(TTextObj)
  private
    FTexts: TStrings;
    procedure SetTexts(const Value: TStrings);
```

```
  public
    constructor Create(AOwner: TComponent); overload; override;
    Destructor Destroy; override;
    procedure Save(f: TIniFile; Sec: String); override;
    procedure Load(f: TIniFile; Sec: String; List: TStrings);
        override;
    property Texts: TStrings read FTexts write SetTexts;
  end;
```

В класі `TTextPoint` визначено:

- `Dx, Dy: Integer` – поля для зберігання координат точки до якої відноситься підпис;
- `FWith, FHeight: Integer` – поля відстаней від точки до лівого верхнього кута напису по осях OX та OY, відповідно. Дані відстані використовуються для збереження взаємного розташування точки та її підпису під час переміщення точки по графічній області;
- метод `GraphCaption` для формування та відображення підпису;

Клас `TTextPoint` містить підмінений метод `LabelMouseMove`, в якому передбачено можливість переміщення підпису лише в межах деякого кола радіусом (15 пікселів) відносно точки, до якої відноситься підпис. Якщо напис знаходиться в межах визначеного радіусу то його переміщення буде відбуватися за рахунок методу батьківського, інакше переміщення не буде.

Зважаючи на те що текстовий напис для пояснень може містити не один рядок в класі `TTextGraph` визначено поле `Texts: TStrings` для зберігання набору рядків.

Студентам можна запропонувати наступне завдання.

Завдання 5 (ППЗ Gran2d).

Обов'язковий рівень. Розробити класи `TTextObj` та `TTextPoint`, розмістивши їх в модулі `UTextObj`:

```
  TTextObj=class(TLabel)
  private
    XOld, YOld: Integer;
    FlagMove: Boolean;
```

```
    procedure LabelMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);virtual;
    procedure LabelMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);virtual;
    procedure LabelMouseMove(Sender: TObject; Shift: TShiftState;
      X,Y: Integer); virtual;
  public
    GrPlace: TGrPlace;
    constructor Create(AOwner: TComponent); overload; override;
    constructor Create(AOwner: TComponent; APlace: TGrPlace);
      overload;
    procedure SetPopMenu(APopupMenu: TPopupMenu);
    procedure Save(f: TIniFile; Sec: String); virtual; abstract;
    procedure Load(f: TIniFile; Sec: String; List: TStrings);
      virtual; abstract;
  end;

  TTextPoint=class(TTextObj)
  public
    Dx, Dy : Integer;
    FWith, FHeight : Integer;
    procedure GraphCaption(const Value: String; AColor: TColor;
        ASize: Integer; AName: String);
    procedure LabelMouseMove(Sender: TObject; Shift: TShiftState; X,
        Y: Integer);override;
  end;
```

Внести зміни в клас `TpointObj`: змінити тип поля `Dlab` на `TTextPoint`; довизначити конструктор в якому передбачити створення підпису; доповнити методи `SetDGx` та `SetDGy` операторами для задання положення підпису та внесення в поля підпису необхідних значень; внести зміни в метод відображення точки з врахуванням зміни типу підпису.

Підвищений рівень. Виконати завдання обов’язкового рівня та доповнити клас `TpointObj` необхідними полями для можливості задання характеристик підпису та внести необхідні налагодження в вікно створення точки. Доповнити

класи `TTextObj` та `TTextPoint`: доповнити методами `Save` та `Load` – для збереження і завантаження властивостей створюваних об'єктів за рахунок використання текстових файлів спеціального виду (ini-файлів).

Перерахунок залежних об'єктів

За ППЗ має забезпечуватися не лише створення об'єктів, а і можливість внесення змін у вже створювані об'єкти. Зважаючи на те, що в ППЗ Gran2d деякі об'єкти є залежними від інших, виникає необхідність в перерахунку певних характеристик залежного об'єкту при зміні певних характеристик базового об'єкту. Для перерахунку деякого одного об'єкту, за його номером в списку об'єктів, описано підпрограму `Procedure ReCalcActObj(n: Integer)`. В даній підпрограмі спочатку відбувається виклик перерахунку характеристик базових об'єктів для даного, а потім перераховуються характеристики самого даного об'єкту.

Для перерахунку всіх об'єктів передбачено процедуру `ReCalcObj`.

Студентам можна запропонувати наступне завдання для вдосконалення створеного ППЗ.

Завдання 6 (ППЗ Gran2d).

Обов'язковий рівень. Доповнити програму створену в попередньому завданні послугами: зміни раніше створених об'єктів; виклику індивідуального контекстного меню для об'єктів розміщених в графічній області.

Підвищений рівень. Виконати завдання обов'язкового рівня та доповнити програму послугою переміщення об'єктів "точка" по графічній області при утримуванні на ній лівої кнопки миші.

2.4.4. Призначення та особливості інтерфейсу ППЗ Numet. Однією з областей математики, де використання засобів обчислювальної техніки вбачається особливо ефективним, є чисельні методи, тому на кафедрі інформатики і ОТ Чернігівського національного педагогічного університету імені Т.Г.Шевченка Костюченком А.О. під керівництвом Горошка Ю.В. було розроблено ППЗ Numet, для підтримки навчання чисельних методів математики.

В ППЗ розглянуто використання наступних чисельних методів математики:

- знаходження наближеного кореня рівняння методом поділу відрізка навпіл, методом хорд, методом дотичних, комбінованим методом;
- знаходження наближеного значення визначеного інтегралу методом Сімпсона, методом трапецій, методом лівих, правих та центральних прямокутників;
- знаходження розв'язку системи лінійних алгебраїчних рівнянь (СЛАР) методом Гаусса, методом Крамера, методом ітерацій, методом Зейделя;
- побудова інтерполяційного многочлена Лагранжа або Ньютона;
- розв'язання звичайних диференціальних рівнянь з заданим початковим значенням методом Ейлера, удосконаленим методом Ейлера, методом Ейлера-Коші.

Для забезпечення виконання зазначених чисельних методів в програмі "Numet" передбачено можливості використання декількох математичних об'єктів:

- залежностей між змінними *x* і *y*, заданими явно у вигляді $y = f(x)$;
- залежностей для задання звичайних диференціальних рівнянь у вигляді $y'(x) = f(x, y)$;
- коефіцієнтів СЛАР;
- таблично заданих функцій, для яких будується інтерполяційний многочлен.

Так само, як і в ППЗ Gran2d, інтерфейс ППЗ Numet базується на MDI-стандарті. Крім головного вікна яке містить головне меню для доступу до всіх послуг програми, ППЗ має 2 додаткові вікна (список об'єктів та розв'язання), розмір і положення яких можна змінювати в межах головного вікна (рис. 2.13).

У вікні *"Список об'єктів"* є список для вибору типу об'єкта, що буде створюватися, список введених об'єктів, панель виведення даних про поточний об'єкт (об'єкт, виділений кольором у списку об'єктів). Операції за програмою виконуються над поточним об'єктом списку об'єктів.

У вікні *"Розв'язання"* виводиться умова виконуваного завдання та хід його розв'язання з згенерованого HTML документу.

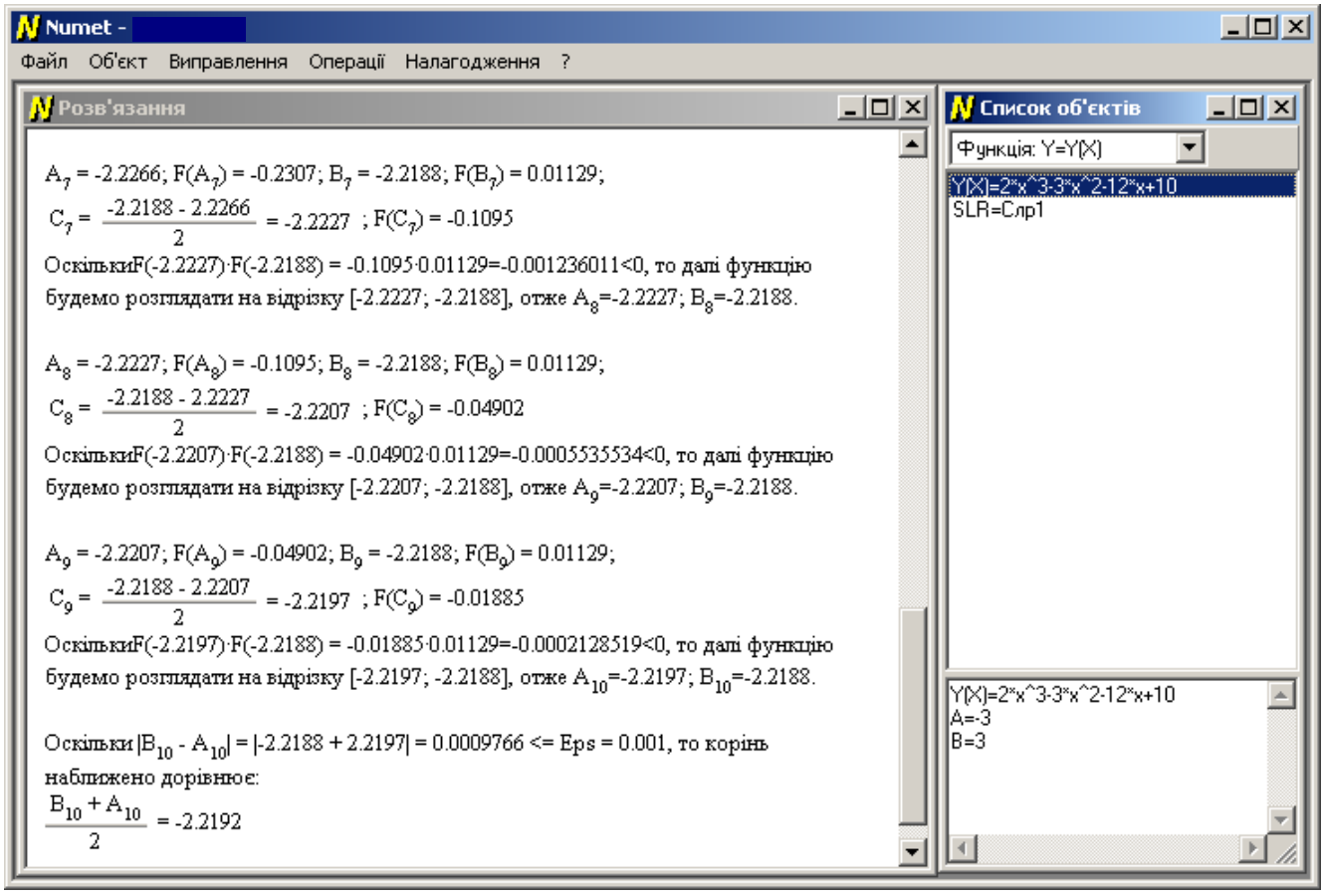


Рис. 2.13. Головне вікно програми Numet.

Для встановлення загальних параметрів роботи з програмою використовується меню “*Налагодження*”. При налагодженні параметрів програми можна встановити:

- мову інтерфейсу;
- розмір шрифту текстових даних, які будуть виводитися в вікні *“Розв’язання”*;
- мінімальну кількість знаків після коми, які будуть враховуватися в процесі обчислень та при виведенні цих обчислень (*“відносно вказаної точності”* – кількість знаків після коми залежить від кількості знаків вказаної точності при виконанні обчислень, *“за точністю даних”* – кількість знаків після коми залежить від кількості знаків в вхідних числових даних, *“рівною вказаній”* – кількість знаків після коми визначається конкретно заданим числом);
- необхідність запам’ятовування параметрів поточної сесії програми при виході з неї.

Що стосується мови інтерфейсу, то програма має український інтерфейс, причому підтримку інших мов легко додати шляхом створення текстових файлів, що містять значення елементів інтерфейсу та повідомлень програми потрібною мовою. Розширення імені такого файлу повинне бути LNG, він повинен знаходитися у папці програми.

ППЗ Numet містить набір допоміжних вікон для створення окремих об'єктів. Для створення нового об'єкту необхідно встановити його тип (обравши тип у списку типів, що знаходиться у верхній частині вікна *"Список об'єктів"*) і звернутися до послуги *"Об'єкт/Створити"*. Після цього на екран виводиться відповідне до типу створюваного об'єкту допоміжне вікно. Також можна скористатися командою *"Створити"* із контекстного меню вікна *"Список об'єктів"*.

Для редагування створених об'єктів (тобто зміни їх характеристик) використовують команду *"Змінити"* з контекстного меню вікна *"Список об'єктів"* чи контекстного меню відповідного об'єкту в списку об'єктів. При цьому на екран викликається те саме допоміжне вікно, яке використовувалася при створенні об'єкту.

Для вилучення об'єкту використовують команду *"Вилучити"* з контекстного меню вікна *"Список об'єктів"*.

Для запису у файл виразів всіх математичних об'єктів із списку об'єктів використовують послугу *"Файл/Записати дані"*. Якщо збереження відбувається перший раз, за програмою запитується ім'я файлу. Для збереження об'єктів у файл з новим іменем використовують послугу *"Файл/Записати дані як…"***.** За допомогою послуги *"Файл/Завантажити…"* можна завантажити до програми об'єкти, збережені у файлі, при цьому всі попередні об'єкти із програми будуть вилучені, і якщо вони не були збережені у файлі, буде запропоновано зберегти їх. Для додавання до існуючого списку об'єктів з файлу використовують послугу *"Файл/Додати до існуючих…"*. За замовчуванням файли, створені за програмою Numet, мають розширення *NUM*. Окрім збереження математичних об'єктів передбачено збереження і проведених обчислень використовуючи послуги

“Файл/Записати розв’язки ” та *“Файл/Записати розв’язки як…”*. Файли розв’язків мають розширення HTM, оскільки дані, що в них містяться записані в HTML форматі.

Доступ до операцій, які можна виконувати над математичними об’єктами, здійснюється через меню *“Операції”* чи через контекстне меню вікна *“Список об’єктів”*. Тип можливої операції залежить від типу поточного об’єкту. Окрім можливих операцій над поточним об’єктом за рахунок ППЗ Numet можна знаходити значення похідної до функції в аналітичному вигляді, для цього необхідно скористатися послугою головного меню *“Операції/Пошук похідної”*.

Для розгляду функціональних можливостей, які забезпечуються за рахунок використання ППЗ Numet розглянемо ряд прикладів по застосуванню ППЗ.

Приклад 2.1. Уточнити корені рівняння $2x^3 - 3x^2 - 12x + 10 = 0$, на відрізках *[-3; -2]* та *[2; 3]* з точністю *0.001*.

Розв’язання. Оскільки задане рівняння від однієї змінної то для створення відповідного об’єкту в списку типів об’єктів вікна *“Список об’єктів”* вибираємо тип *“Функція: Y=Y(X)”*. Далі скориставшись послугою головного меню *“Об’єкт/Створити”* в відкритому додатковому вікні (рис. 2.14) вводимо рівняння та відрізок задання, який буде включати обидва відрізки на яких мають уточнюватися корені рівняння і підтверджуємо введення характеристик створюваного об’єкту.

В результаті зазначених дій в списку об’єктів буде додано об’єкт, який відповідатиме нашому рівнянню. Далі можна переходити до уточнення кореня. Скориставшись послугою головного меню *“Операції/Розв’язати рівняння”* обираємо один з можливих методів, за рахунок якого і буде відбуватися уточнення кореня. Після виконаного вибору в відкритому додатковому вікні (рис. 2.15) необхідно вказати межі відрізку на якому буде відбуватися уточнення кореня та точність яка, буде вважатися достатньою для завершення уточнення та підтвердити задані параметри.

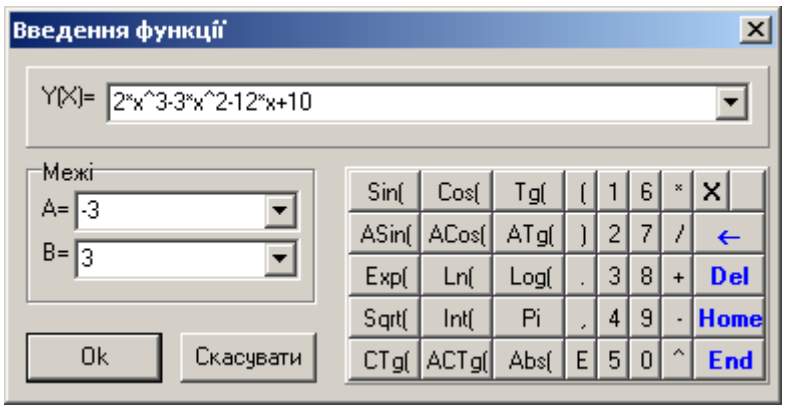


Рис. 2.14. Вікно задання характеристик функціональній залежності.

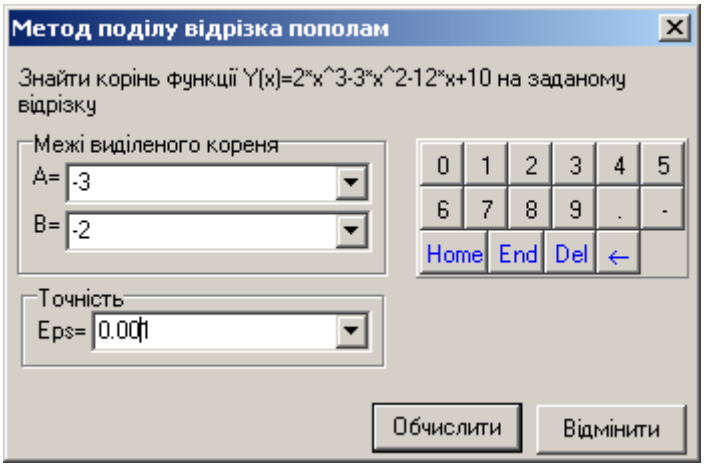


Рис. 2.15. Вікно задання характеристик для застосування методу пошуку кореня рівняння

В результаті проведених обчислень у вікні *"Розв'язання"* з'явиться розв'язок нашого завдання (див. рис. 2.13), а саме: спочатку буде виведена умова завдання, потім хід розв'язання з виведенням пояснень всіх кроків щодо уточнення кореня рівняння і виведено остаточний результат. У випадку, якщо обраний нами метод не може бути застосований до розв'язання конкретного рівняння, відповідне пояснення щодо неможливості застосування цього методу, також буде відображуватися в вікні *"Розв'язання"*. Аналогічно виконуються дії по уточненню кореня рівняння і на іншому відрізку.

Також введену функціональну залежність, можна використати і для знаходження наближеного значення визначеного інтегралу, наприклад, як у наступному прикладі.

Приклад 2.2. Обчислити значення визначеного інтегралу для функції $F(x) = 2x^3 - 3x^2 - 12x + 10$, на відрізках *[2; 4]* розбивши його на 10 частин.

Розв’язання. Оскільки створення даної функції аналогічне до створення функції в попередньому прикладі, то даний опис буде опущено. Після створення функції, скориставшись послугою головного меню *“Операції/Знайти значення інтегралу”* і вибравши метод, за рахунок якого буде відбуватися обчислення в відкритому додатковому вікні, необхідно задати характеристики для застосування відповідного методу (відрізок інтегрування та степінь розбиття відрізку) (рис. 2.16).

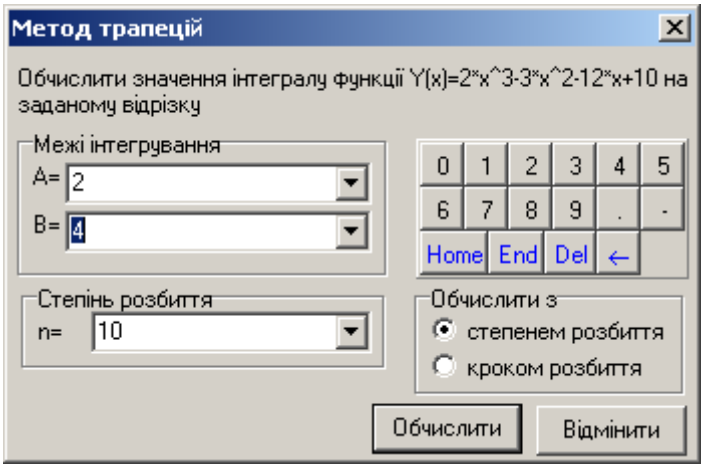


Рис. 2.16. Вікно задання характеристик для застосування методу обчислення значення визначеного інтегралу

Варто відмітити, що інтегрування може бути проведене не лише заданням степеня інтегрування (кількості відрізків розбиття), а і заданням кроку розбиття, таким чином кількість відрізків розбиття буде обраховано автоматично. В результаті проведених обчислень користувачу у вікні *“Розв’язання”* буде відображено і умову задачі і хід розв’язання з покроковим виведенням результату (рис. 2.17).

Приклад 2.3. Знайти розв’язок системи лінійних алгебраїчних рівнянь з точністю 0.001.

$$\begin{cases} 2.173X_1 - 0.357X_2 + 1.077X_3 = 1.245 \\ 0.453X_1 - 3.986X_2 + 0.988X_3 = 3.517 \\ 0.427X_1 + 0.770X_2 + 4.457X_3 = 1.273 \end{cases}$$

Розв’язання. Перш за все необхідно створити об’єкт, який буде відповідати заданій СЛАР. Для цього у вікні *“Список об’єктів”* вибираємо тип створюваної функції *“Таблиця СЛАР:”*. Далі, скориставшись послугою головного меню

"Об'єкт/Створити" у відкритому додатковому вікні (рис. 2.18), використовуючи інструменти вилучення рядків та стовпців, встановлюємо їх кількість, відповідно до нашої СЛАР та вводимо коефіцієнти СЛАР до відповідних комірок (стовпець В є стовпцем вільних членів).

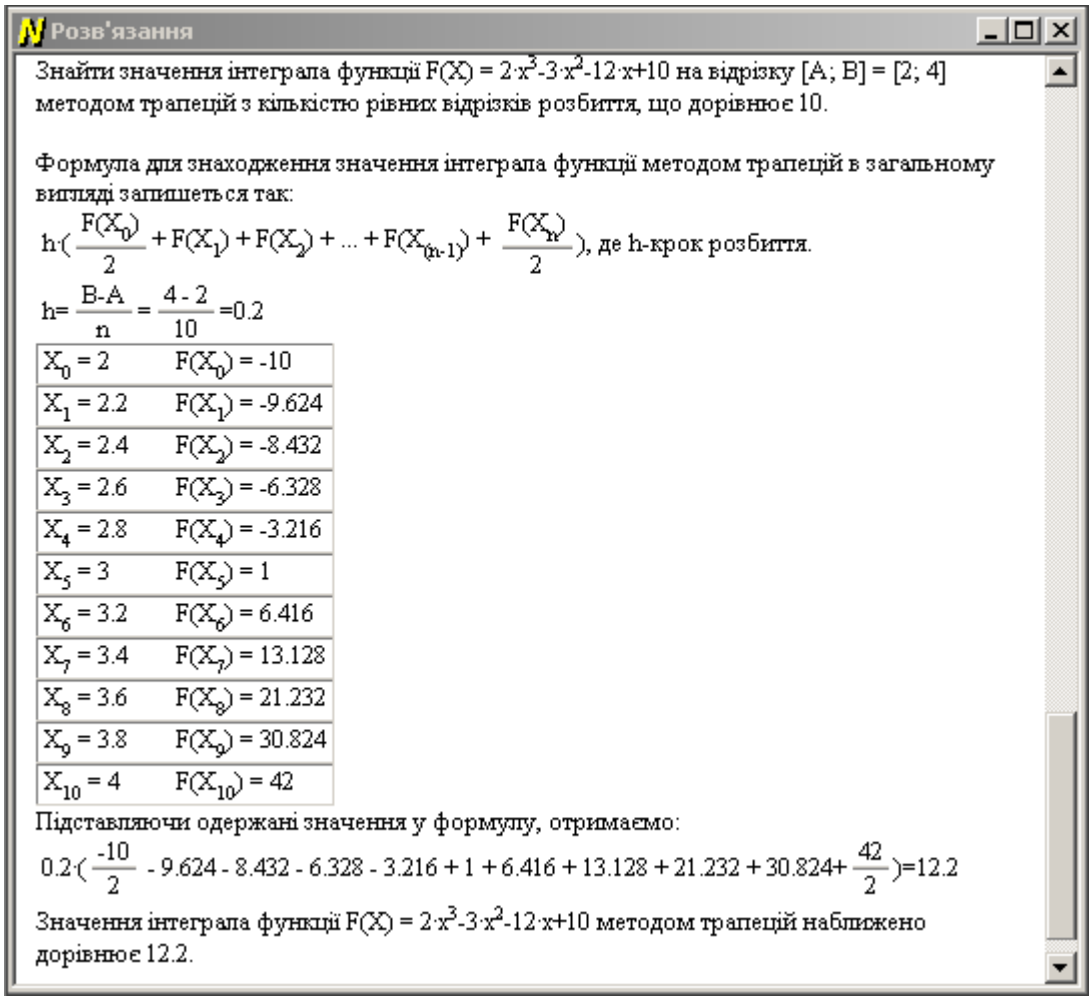


Рис. 2.17. Вікно розв'язання після обчислення значення визначеного інтегралу

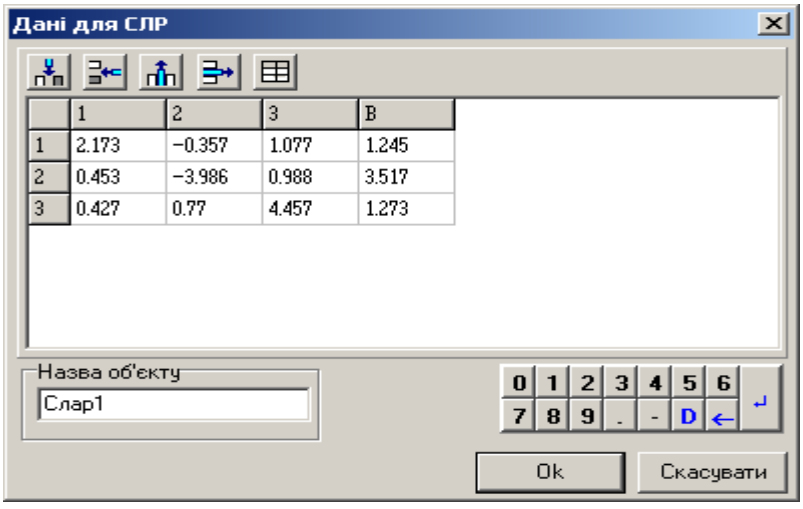
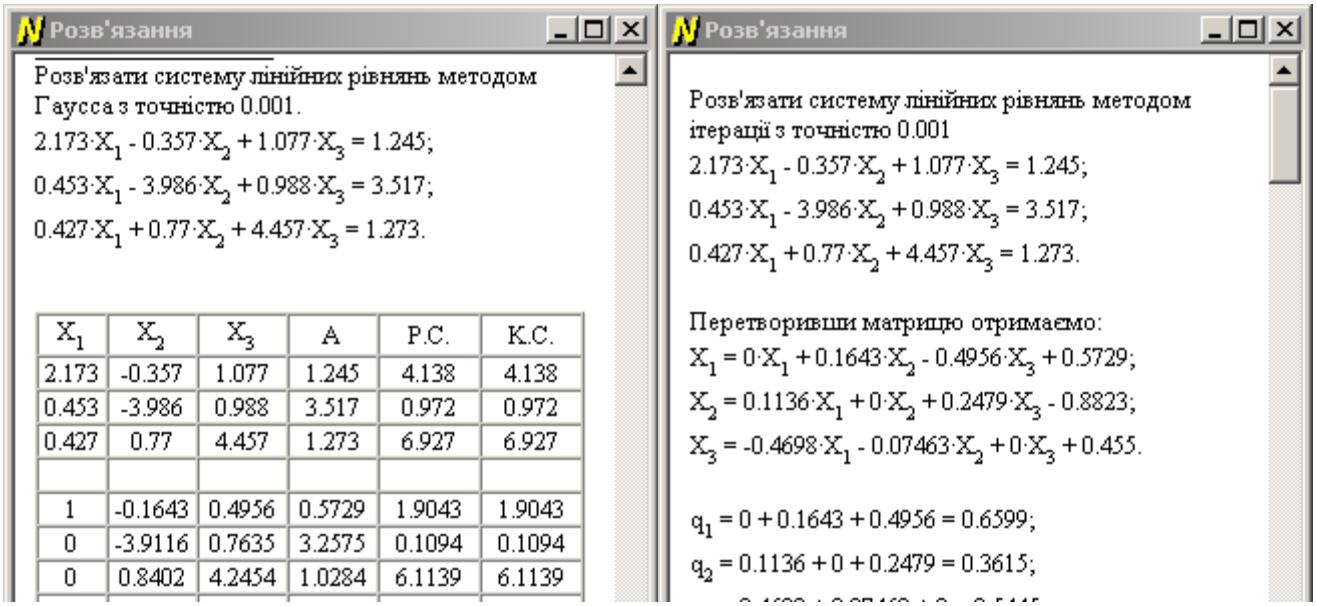


Рис. 2.18. Вікно задання коефіцієнтів СЛАР

Після введення характеристик та створення об’єкту для пошуку розв’язків СЛАР необхідно скористатися послугою головного меню *“Операції/Розв’язати СЛАР” ”* і вибравши метод за рахунок якого буде відбуватися такий пошук, необхідно уточнити точність пошуку. В результаті обчислення у вікні *“Розв’язання”* буде відображено і умову задачі і хід розв’язання з покроковим виведенням результату (рис. 2.19, а).

У випадку застосування методу ітерацій чи Зейделя для пошуку розв’язків СЛАР, вхідна матриця коефіцієнтів спочатку буде перетворена (рис. 2.19, б).



а) метод Гауса б) методом Ітерацій

Рис. 2.19. Вікно розв’язання після знаходження розв’язку СЛАР

2.4.5. Методика навчання розробки ППЗ з чисельних методів математики на прикладі ППЗ Numet.

Ієрархія класів, загальні типи даних та константи. Клас TFunc.

Розглянемо тепер, як здійснюється опис і реалізація основних об’єктів ППЗ Numet, зазначених раніше. Реалізації окремих методів класів, що розглядаються, та деяких підпрограм загального призначення винесені до додатку Е.

Сукупність об’єктів утворюють ієрархію, на вершині якої знаходиться клас `TNumerMet` в якому зосереджені загальні характеристики розглядуваних об’єктів (рис. 2.20). В подальшому специфічні характеристики окремих об’єктів будуть довизначатися в класах нащадках.

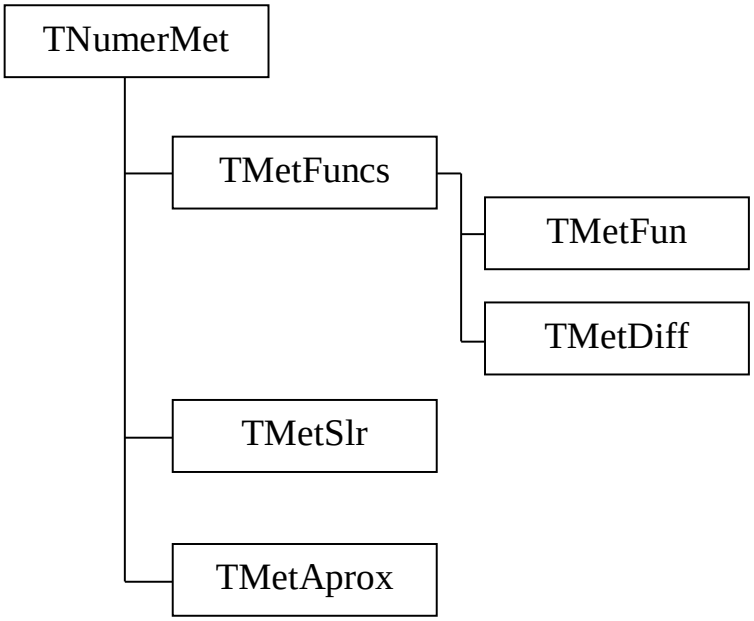


Рис. 2.20. Дерево класів програми Numet

Перш ніж переходити до опису класів, введемо необхідні типи даних

```
TypeFunc=(tfNone, tfNormal, tfNormalY, tfMatrica, tfTablica);
DataArr=Array[0..ElementCount] Of Extended;
DataMas=Array[0..ElementCount, 0..ElementCount] Of Extended;
```

Тип `TypeFunc` визначає тип об’єкту, що буде задаватися: `Normal` – явно задана функція для знаходження наближеного кореня рівняння та наближеного значення визначеного інтегралу; `NormalY` – неявно задана функція для розв’язання звичайних диференціальних рівнянь з заданим початковим значенням; `Matrica` – двовимірна матриця для знаходження розв’язку СЛАР; `Tablica` – таблично задана функція для побудови інтерполяційного многочлена.

Типи `DataArr` та `DataMas` задають одновимірний та двовимірний масиви для зберігання таблично заданої функції, коефіцієнтів СЛАР та допоміжних обчислень. Константа `ElementCount` задає можливу максимальну кількість елементів відповідних масивів.

Для обчислення функціональних залежностей використовується клас `TFunc`. Наведемо фрагмент опису цього класу. Деякі несуттєві для розгляду у даному курсі поля та методи в подальших описах будуть опущені

```
type
  TFunc=class(TObject)
   public
```

```
    Tx, St: String;
    Arr: RealArray;
    constructor Create(SF: String);
    function Deriv(Arg: Extended): Extended;
    function Deriv2(Arg: Extended): Extended;
    procedure ChangeFunc(SF: String; var Chng: Boolean);virtual;
    function Eval(ArgX, ArgY: Extended): Extended;
  End;
```

Батьківським класом для `TFunc` є базовий клас мови Object Pascal `TObject`. Навіть якщо при описі нового класу не вказати батьківський клас, то за замовчуванням ним буде клас `TObject`.

Клас `Tfunc` містить конструктор `Create(SF: String)`. Конструктор – це специфічний метод, призначений для створення і ініціалізації об'єктів класу. Конструктор призначений для виділення пам'яті під об'єкт і ініціалізації його полів. Для створення об'єкту викликають його конструктор певним чином. У класі `TFunc` конструктор `Create` має єдиний параметр – вираз функції. Наведемо приклад створення об'єкту типу `TFunc`, що міститиме вираз `'x'`:

```
Var f: TFunc;
…
f:=TFunc.Create('x');
```

В описі конструктора потрібно спочатку викликати конструктор батьківського класу, а потім виконати додаткові дії, пов'язані з ініціалізацією полів:

```
constructor TFunc.Create(SF: String);
...
Begin
  Inherited Create;
  ...
End;
```

При зверненні до функції

```
function Eval(ArgX, ArgY: Extended): Extended;
```

повертається значення виразу для вказаних значень аргументів. Оскільки в програмі Numet вирази бувають як з однією, так і з двома змінними, при

зверненні до даної функції вказуються значення двох аргументів. Якщо вираз містить тільки одну змінну, значення другого аргументу ігнорується. Якщо у процесі обчислення значення виразу виникла помилка (ділення на нуль, добування квадратного кореня із від'ємного числа тощо), за функцією `Eval()` повернеться значення константи `AllErr=1E+100` за допомогою опрацювання виключень. Це надає можливість аналізувати в подальшому такі помилки.

При зверненні до функції

```
function Deriv(Arg: Extended): Extended;
function Deriv2(Arg: Extended): Extended;
```

повертається значення відповідно першої та другої похідної для вказаного аргументу. Похідні від виразів, що містять дві змінні, у програмі Numet не розглядаються, тому аргумент в цій функції тільки один.

Процедура

```
procedure ChangeFunc(SF: String; var Chng: Boolean); virtual;
```

призначена для зміни значення виразу; `SF: String` – це новий вираз, у параметрі `var Chng: Boolean` повертається значення `True`, якщо зміна відбулася, і `False` в протилежному випадку (зміна може не відбутися, якщо вираз `SF` є некоректним). Ця процедура також є віртуальною, тому що буде змінюватися у нащадках.

Для функціонування класу `TFunc` необхідно підключити модуль NGMath. Даний модуль містить процедури, за якими на основі рядка запису виразу будують певну структуру даних, що використовується у методі `Eval()` для обчислення значення цього виразу. Всі інші необхідні типи та константи буде розглянуто по мірі їх необхідності.

Після розгляду даного матеріалу студентам можна запропонувати наступне завдання.

Завдання 1 (ППЗ Numet).

Обов'язковий рівень. Розробити клас `TFunc`, розмістивши його в модулі `NGFunc` та підключивши до нього модуль `NGMath`.

```
TFunc=class(TObject)
 public
  Tx,St: String;
```

```
    Arr: RealArray;
    constructor Create(SF: String);
    function Deriv(Arg: Extended): Extended;
    procedure ChangeFunc(SF: String; var Chng: Boolean);virtual;
    function Eval(ArgX, ArgY: Extended): Extended;
  End;
```

Написати консольну програму, за якою вводиться значення виразу з клавіатури і відбувається табулювання функції, заданої виразом, на деякому відрізку з кроком 0,5. Для кожного отриманого аргументу під час табулювання, окрім значення функції передбачити обчислення значення першої похідної.

Підвищений рівень. Виконати завдання обов'язкового рівня та доповнити клас `TFunc` методом, за рахунок якого має відбуватися обчислення значення другої похідної для вказаного аргументу.

Базовий клас TNumerMet. Клас TMetFuncs.

Клас `TNumerMet` виступає батьківським класом для класів опрацьовуваних об'єктів в ППЗ Numet. В даному класі згруповані загальні властивості та методи.

Опис даного класу має наступний вигляд:

```
TNumerMet=class(TObject)
  private
    . . .
    Procedure SInpH(StInp: String);
  protected
    function GetTypeName: String; virtual; abstract;
  public
    property Eps :  Extended read FEps write FEps;
    property FuncType : TypeFunc read FFuncType write FfuncType;
    constructor create;
    procedure InputInfo(M: TStrings); virtual; abstract;
    procedure Load(f: TIniFile; Sec: String);
    procedure Save(f: TIniFile; Sec: String); virtual; abstract;
 End;
```

В даному класі визначено такі загальні властивості: кожний створюваний об'єкт належить певному типу `FuncType: TypeFunc`, та містити точність

обчислень, що будуть проводитися `Eps: Extended`. Також додатково введено приватне поле `Eps_c: Integer`, за яким визначається необхідна кількість знаків після коми для виведення виконуваних обчислень. Ця кількість знаків після коми буде залежати від вказаної точності обчислення та кількості знаків після коми в вхідних характеристиках об'єктів (кінців відрізків інтегрування, значень таблиці задання функції, значень коефіцієнтів СЛАР). Обчислення кількості значень після коми будуть проводитися за рахунок перезавантажених функцій `AfterPoint`.

При вхідному єдиному параметрі:

```
function AfterPoint(Data: Extended): Integer; overload;
```

При вхідних трьох параметрах:

```
function AfterPoint(Data1, Data2, Data3: Extended): Integer;
    overload;
```

При вхідному двовимірному масиві, за рахунок якого задається таблично задана функція та кількістю значень в цьому масиві:

```
function AfterPoint(Data: DataArr; kol: Integer): Integer;
    overload;
```

При вхідному двовимірному масиві, за рахунок якого задається матриця коефіцієнтів СЛАР та кількістю рядків і стовпців цієї матриці:

```
function AfterPoint(Data: DataMas; kolr, kolc: Integer): Integer;
    overload;
```

В області `protected` для можливого використання в класах нащадках, що будуть розташовані в інших модулях, описано віртуальний, абстрактний метод:

```
function GetTypeName: String; virtual; abstract;
```

який призначений для повернення текстового запису типу об'єкту. Дана функція використовується при записі об'єкту в файл.

Серед прихованих методів даний клас містить метод:

```
Procedure SInpH(StInp: String);
```

який призначений для виведення текстових даних, які будуть формуватися в процесі обчислень. Таке виведення може проводитися в область виведення вікна *"Розв'язання"* чи деякий проміжний компонент для подальшого виведення чи збереження. На початковому етапі це виведення може проводитися в звичайний

компонент багаторядкового виведення `TMemo`, а в подальшому для отримання більш наочного, з математичної точки зору, результату, виведення може проводитися у форматований HTML документ.

Конструктор класу `TNumerMet` містить оператори виклику конструктору батьківського класу, для виділення пам'яті під створюваний об'єкт та оператори, за допомогою яких будуть надаватися початкові значення полям створюваного об'єкту.

При визначенні методів для класу `TNumerMet` було передбачено можливість роботи з файлами, а саме збереження та завантаження об'єктів. Для цього призначені віртуальні, абстрактні методи, які в нащадках будуть довизначатися в залежності від введення нових характеристик об'єкту.

Збереження створеного об'єкту в файл буде відбуватися за рахунок методу

```
procedure Save(f: TIniFile; Sec: String); virtual; abstract;
```

Для завантаження в середовище програми, раніше збережених об'єктів призначено процедуру

```
procedure Load(f: TIniFile; Sec: String; List: TStrings);
```

Дані будуть зберігатися в структурованому файлі типу `TIniFile`, тому обидві процедури містять параметр `Sec: String`, за яким буде задаватися текстова назва розділу в ini файлі. Процедура `Save` в класах нащадках підмінятиметься в залежності від характеристик того чи іншого класу, а процедура `Load` є відразу визначеною для можливості завантаження об'єктів різних типів і може бути доповнена при розширенні дерева класів.

Під час роботи програми створювані об'єкти будуть зберігатися в візуальному списку класу `TListBox`, тому метод завантаження містить параметр `List`, за яким і буде визначатися список, в якому мають зберігатися об'єкти. Реалізація процедури завантаження має наступний вигляд:

Крім того, в класі `TNumerMet` визначено абстрактний, віртуальний метод `InputInfo(M: TStrings); virtual; abstract`; за яким до параметру `M: TStrings` будуть заноситися характеристики об'єкту. Даний метод

використовується для виведення характеристик про поточний об’єкт в вікні “*Список об’єктів*”.

Отже після того, як буде описаний батьківський клас з визначеними загальними властивостями, можна приступати до опису похідних класів, тобто класів-нащадків, які будуть описувати новий об’єкт з зазначеної ієрархії.

Наступним класом для розгляду є клас `TMetFuncs`, який є проміжним для класів, призначених для опрацювання залежностей виду $y = f(x)$ та $y'(x) = f(x, y)$.

```
TMetFuncs=class(TNumerMet)
  private
   f   : TFunc;
   . . . { опис полів та методів згідно властивостей }
  public
   property Func : String read FFunc write FFunc;
   property Xa : Extended read FXa write FXa;
   property Xb : Extended read FXb write FXb;
   property Xot : Extended read FXot write FXot;
   property Xdo : Extended read FXdo write FXdo;
   property h : Extended read Fh write Fh;
   constructor create;
   procedure InputInfo(M: TStrings); override;
   procedure Save(f : TIniFile; Sec : String); override;
 end;
```

У цьому класі описані властивості, через які будуть задаватися загальні характеристики функції.

Даний клас містить поле `f: TFunc` за яким буде відбуватися опрацювання функціональної залежності заданої в текстовому вигляді.

За властивістю

```
property Func : String read FFunc write FFunc;
```

визначається рядковий запис функціональної залежності.

За властивостями:

```
property Xa : Extended read FXa write FXa;
```

```
property Xb : Extended read FXb write FXb;
```

визначаються початок та кінець відрізку задання функції.

За властивостями:

```
property Xot : Extended read FXot write FXot;
property Xdo : Extended read FXdo write FXdo;
```

визначаються початок та кінець відрізку на якому буде проводитися обчислення. Дані властивості введені для виконання обчислень на різних відрізках, відмінних від відрізку задання функції.

За властивістю

```
property h : Extended read Fh write Fh;
```

визначається крок розбиття відрізку для проведення обчислення.

Окрім того в даному класі необхідно підмінити деякі батьківські методи, а саме: метод `InputInfo`, за рахунок якого відбувається формування характеристичних даних про об'єкт та метод `Save(f: TIniFile; Sec: String)`, за рахунок якого відбувається збереження даних про об'єкт в секцію `Sec` ini файлу пов'язаного з файловою змінною `f`.

Студентам можна запропонувати наступне завдання.

Завдання 2 (ППЗ Numet).

Обов'язковий рівень. Розробити класи `TNumerMet` та `TMetFuncs`, розмістивши їх в модулі `NGFuncs`.

```
TNumerMet=class(TObject)
  private
   FEps :  Extended;
   FFuncType : TypeFunc
   Procedure SInpH(StInp: String);
  protected
   function GetTypeName: String; virtual; abstract;
  public
   constructor create;
   procedure InputInfo(M: TStrings); virtual; abstract;
   property Eps :  Extended read FEps write FEps;
   property FuncType : TypeFunc read FFuncType write FfuncType;
```

```
 End;
TMetFuncs=class(TNumerMet)
  private
   f   : TFunc;
   . . . { опис полів та методів згідно властивостей }
  public
   constructor create;
   procedure InputInfo(M: TStrings); override;
   property Func : String read FFunc write FFunc;
   property Xa : Extended read FXa write FXa;
   property Xb : Extended read FXb write FXb;
   property Xot : Extended read FXot write FXot;
   property Xdo : Extended read FXdo write FXdo;
   property h : Extended read Fh write Fh;
 end;
```

Створити програму з графічним інтерфейсом, в якій передбачити можливість створення декількох екземплярів класу `TMetFuncs` з розміщенням їх в списку візуального компонента `TListBox`. При виділенні створеного об’єкту в списку, дані про його характеристики мають виводитися в компоненту багаторядкового виведення. Також описати підпрограму, за якою для виділеного об’єкту буде відбувається табулювання відповідної функції, заданої цим об’єктом. Табулювання повинно відбуватися на певному відрізку та з певним кроком, які користувач має задати через допоміжне вікно. Для кожного отриманого аргументу під час табулювання, окрім значення функції, передбачити обчислення значення першої похідної.

Підвищений рівень. Виконати завдання обов’язкового рівня та доповнити клас `TMetFuncs` методом `Save` за рахунок якого має відбуватися збереження властивостей об’єкту в текстовий файл спеціального виду (ini-файл) та клас `TNumerMet` методом `Load` за рахунок якого має відбуватися завантаження екземплярів класу `TMetFuncs` з наперед збереженого файлу спеціального виду (ini-файл) до списку об’єктів.

Клас для задання функціональних залежностей у явному вигляді $y = f(x)$**.**

Клас для задання звичайних диференціальних рівнянь у вигляді $y'(x) = f(x, y)$**.**

Наступним розглянутим класом буде клас `TMetFun` для опрацювання явно заданої функції в вигляді $y = f(x)$ і виконання чисельних методів знаходження наближеного кореня рівняння та обчислення наближеного значення визначеного інтегралу.

Опис класу `TMetFun` наступний:

```
TMetFun=class(TMetFuncs)
  private
   . . . { опис полів та методів згідно властивостей }
   function GetTypeName: String; override;
   procedure StartUmovu;
  public
   property StepinRozb : Boolean read FStepinRozb write
      FStepinRozb;
   property VRoz : Integer read FVRoz write FVRoz;
   Procedure MPopolam;
   Procedure MXord;
   Procedure MDotuchnux;
   Procedure MKombi;
   Procedure MTrapeciy;
   Procedure MSimpson;
   Procedure MPrymLeft;
   Procedure MPrymRigth;
   Procedure MPrymCenter;
 end;
```

Даний клас містить поля, які будуть необхідні для виконання зазначених обчислень та поля, які будуть характеризувати його екземпляри. Так за полем `VRoz: Integer` буде визначатися кількість відрізків розбиття, а за полем `StepinRozb: Boolean` буде визначатися тип задання розбиття (коли розбиття вказується кількістю відрізків розбиття та потребує розрахунку кроку розбиття то

поле буде містити `True`, коли розбиття вже вказане кроком розбиття, то поле буде містити `False`).

Окрім того клас `TMetFun` містить метод, за яким буде визначатися тип об'єкту в текстовому вигляді `GetTypeName: String`, який буде використовуватися в методі збереження та метод `procedure StartUmovu`, за яким буде відбуватися задання початкових умов для виконання обчислень, зокрема визначення необхідної кількості знаків після коми для виведення числових результатів.

```
procedure TMetFun.StartUmovu;
Begin
  . . .
  Eps_c:=AfterPoint(Eps, Xot, Xdo);
End;
```

Крім того даний клас містить методи, за рахунок яких будуть виконуватися обчислення за відповідними чисельними методами. Для знаходження наближеного кореня рівняння: `MPopolam` – за методом поділу відрізка навпіл, `MXord` – за методом хорд, `MDotuchnux` – за методом дотичних, `MKombi` – за комбінованим методом. Для обчислення наближеного значення визначеного інтегралу: `MTrapeciy` – за методом трапецій, `MSimpson` – за методом Сімпсона, `MPrymLeft` – за методом лівих прямокутників, `MPrymRigth` – за методом правих прямокутників, `MPrymCenter` – за методом центральних прямокутників.

Наступним розглянутим класом буде клас `TMetDiff` для розв'язання звичайних диференціальних рівнянь заданих в вигляді $y'(x) = f(x, y)$ з заданим початковим значенням.

```
TMetDiff=class(TMetFuncs)
 private
  . . . { опис полів та методів згідно властивостей }
  function GetTypeName: String; override;
  procedure StartUmovu;
 public
  property Xpr : Extended read FXpr write FXpr;
  constructor create;
```

```
    Procedure MEylor;
    Procedure MEylKosh;
    Procedure MUdEyl;
  end;
```

Даний клас містить єдине унікальне поле `Xpr: Extended`, за яким задається початкове значення функції в точці `Xa`. Серед методів визначені методи, за якими буде відбуватися розв'язання звичайних диференціальних рівнянь з заданим початковим значенням: `MEylor` –методом Ейлера, `MEylKosh` –методом Ейлера-Коші, `MUdEyl` –удосконаленим методом Ейлера.

Завдання 3 (ППЗ Numet).

Обов'язковий рівень. Розробити клас `TMetFun` розмістивши його в модулі `NGFun` та клас `TMetDiff` розмістивши його в модулі `NGDiff`.

```
TMetFun=class(TMetFuncs)
  private
   FStepinRozb: Boolean;
   FVRoz: Integer;
   function GetTypeName: String; override;
   procedure StartUmovu;
  public
   property StepinRozb : Boolean read FStepinRozb write
      FStepinRozb;
   property VRoz : Integer read FVRoz write FVRoz;
   . . . { обрані чисельні методи відповідно до класу }
 end;
TMetDiff=class(TMetFuncs)
  private
   FXpr: Extended;
   function GetTypeName: String; override;
   procedure StartUmovu;
  public
   property Xpr : Extended read FXpr write FXpr;
   constructor create;
   . . . { обрані чисельні методи відповідно до класу }
 end;
```

Доповнити програму, створену в попередньому завданні додавши до неї послугу створення екземплярів класів `TMetFun`, `TMetDiff` та розміщення їх в візуальному списку об’єктів. В класі `TMetFun` передбачити можливість обчислення наближеного кореня рівняння одним з зазначених методів (методами поділу відрізка навпіл, методом хорд, методом дотичних) та обчислення наближеного значення визначеного інтегралу одним з зазначених методів (методами лівих правих та середніх прямокутників). В класі `TMetDiff` передбачити можливість розв’язання звичайних диференціальних рівнянь одним з зазначених методів (методом Ейлера та удосконаленим методом Ейлера).

Підвищений рівень. Виконати завдання обов’язкового рівня. Доповнити клас `TMetFun` методами за рахунок яких має відбуватися обчислення наближеного кореня рівняння комбінованим методом та обчислення наближеного значення визначеного інтегралу методом Сімпсана або методом трапецій. Доповнити клас `TMetDiff` методом за рахунок якого має відбуватися розв’язання звичайного диференціального рівнянь методом Ейлера-Коші.

Доповнити метод `Load` класу `TNumerMet` операторами за рахунок яких має відбуватися завантаження екземплярів класів `TMetFun` та `TMetDiff` з наперед збереженого файлу спеціального виду (ini-файл) до списку об’єктів.

Клас для задання коефіцієнтів СЛАР.

Клас `TMetSlr` призначений для знаходження розв’язків СЛАР, заданої своїми коефіцієнтами.

Опис цього класу має наступний вигляд:

```
TMetSlr=class(TNumerMet)
  private
   . . . { опис полів та методів згідно властивостей }
   MV: DataArr;
   function GetTypeName: String; override;
   procedure Save(f: TIniFile; Sec: String); override;
   procedure StartUmovu;
   Function Sumistnist: TypeSumist;
   Procedure PeretvorMat(Var m: MasTab; kol: Integer);
```

```
public
 MT: DataMas;
 property Name: String read FName write FName;
 property Kolrow: Integer read FKolrow write FKolrow;
 property Kolcol: Integer read FKolcol write FKolcol;
 property FlagPeretvor: Boolean read FFlagPeretvor write
    FFlagPeretvor;
 procedure InputInfo(M: TStrings); override;
 Procedure MGaus;
 Procedure MKramer;
 Procedure MIterac;
 Procedure MZeydel;
end;
```

Даний клас містить поля: `MT: DataMas` для збереження коефіцієнтів СЛАР та `MV: DataArr` для збереження знайдених коренів. Варто відмітити, що поле `MT` є публічним і буде доступне з інших модулів, оскільки задання коефіцієнтів СЛАР буде відбуватися за рахунок використання додаткового вікна, а поле `MV` є приватним, оскільки воно буде використовуватися лише в межах даного класу.

За властивостями

```
property Kolrow: Integer read FKolrow write FKolrow;
property Kolcol: Integer read FKolcol write FKolcol;
```

визначається кількість рівнянь та кількість невідомих в заданій СЛАР.

Враховуючи те, що для створеного екземпляру класу `TMetSlr` назва, яка буде відображатися в списку об’єктів, не може бути однозначно згенерованою, як це було для екземплярів класів явно та неявно заданих функцій, то для задання такої назви введена властивість

```
property Name : String read FName write FName.
```

СЛАР може представлятися у декілька способів. Так для пошуку розв’язків методами Крамера та Гаусса, СЛАР має задаватися у вигляді:

$$\begin{cases} a_{11}x_1 & +a_{12}x_2 & \cdots & +a_{1n}x_n & = b_1 \\ a_{21}x_1 & +a_{22}x_2 & \cdots & +a_{2n}x_n & = b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1}x_1 & +a_{n2}x_2 & \cdots & +a_{nn}x_n & = b_n \end{cases}$$

Для застосування методів ітерацій та Зейделя необхідно, щоб СЛАР була задана через змінні, наступним чином:

$$\begin{cases} x_1 = & 0 & +a_{12}x_2 & \cdots & +a_{1n}x_n & +b_1 \\ x_2 = & a_{21}x_1 & +0 & \cdots & +a_{2n}x_n & +b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n = & a_{n1}x_1 & a_{n2}x_2 & \cdots & +0 & +b_n \end{cases}$$

Тому в класі введено властивість

```
property FlagPeretvor: Boolean read FFlagPeretvor write
FFlagPeretvor;
```

за рахунок якої буде визначатися чи є необхідність у перетворенні СЛАР через змінні (якщо властивість буде містити `True`, то перед застосуванням методів ітерацій чи Зейделя є необхідність у перерахунку заданих коефіцієнтів для отримання необхідного задання СЛАР, якщо ж властивість буде містити `False` то таке перетворення непотрібне).

Для виконання зазначеного перетворення передбачено метод

```
Procedure PeretvorMat(Var m: MasTab; kol: Integer)
```

за рахунок якого і буде відбуватися добір елементів для можливості перетворення. Такий добір елементів може відбуватися утворенням комбінацій сум та різниць окремих рівнянь системи.

Для СЛАР, заданої своїми коефіцієнтами, може виникнути випадок коли, вона буде нерозв’язною зазначеними чисельними методами. Це може бути у випадку, коли ранги розширеної та звичайної матриць відрізняються, то відповідно система є несумісною і не має розв’язків або коли серед рівнянь є лінійно залежні, то відповідно система має безліч розв’язків. Для перевірки відповідних умов в класі `TMetSlr` передбачено метод `Function Sumistnist: TypeSumist`.

Тип `TypeSumist=(FLZaleg, FRang, FRozv)` характеризує різні випадки сумісності СЛАР: `FLZaleg` – серед рівнянь є лінійно залежні, `FRang` - ранги розширеної та звичайної матриць неоднакові, `FRozv` – система може бути розв'язана.

Також даний клас містить методи для виконання відповідних методів знаходження розв'язку СЛАР, зокрема: `MGaus` – за методом Гаусса, `MKramer` – за методом Крамера, `MIterac` – за методом ітерацій, `MZeydel` – за методом Зейделя.

Завдання 4 (ППЗ Numet).

Обов'язковий рівень. Розробити клас `TMetSlr`, розмістивши його в модулі `NGSlr`.

```
TMetSlr=class(TNumerMet)
  private
   . . . { опис полів та методів згідно властивостей }
   MV: DataArr;
   function GetTypeName: String; override;
   procedure StartUmovu;
   Function Sumistnist: TypeSumist;
   Procedure PeretvorMat(Var m: MasTab; kol: Integer);
  public
   MT: DataMas;
   property Name: String read FName write FName;
   property Kolrow: Integer read FKolrow write FKolrow;
   property Kolcol: Integer read FKolcol write FKolcol;
   property FlagPeretvor: Boolean read FFlagPeretvor write
      FFlagPeretvor;
   procedure InputInfo(M: TStrings); override;
   . . . { обрані чисельні методи відповідно до класу }
end;
```

Доповнити програму, створену в попередньому завданні, додавши до неї послугу створення екземплярів класу `TMetSlr` та розміщення їх у візуальному списку об'єктів. Передбачити можливість знаходження розв'язку системи лінійних алгебраїчних рівнянь методом Гаусса або методом Крамера.

Підвищений рівень. Виконати завдання обов'язкового рівня та доповнити клас `TMetSlr` методом `Save`, за рахунок якого має відбуватися збереження властивостей об'єкту в текстовий файл спеціального виду (ini-файл). Доповнити метод `Load` класу `TNumerMet` операторами, за рахунок яких має відбуватися завантаження екземплярів класу `TMetSlr` з наперед збереженого файлу спеціального виду (ini-файл) до списку об'єктів. Передбачити за рахунок класу `TMetSlr` можливість знаходження розв'язку системи лінійних алгебраїчних рівнянь методом ітерації або методом Зейделя.

Клас для таблично заданої функції.

Клас `TMetAprox` призначений для задання функції в табличному вигляді, побудови інтерполяційного многочленна та обчислення значення многочлена для деякого заданого аргументу. Опис класу має вигляд:

```
TMetAprox=class(TNumerMet)
  private
   . . . { опис полів та методів згідно властивостей }
   MT: DataMas;
   function GetTypeName: String; override;
   procedure StartUmovu;
  public
   MV: DataArr;
   property Name: String read FName write FName;
   property Xpr: Extended read FXpr write FXpr;
   property Kolz: Integer read FKolz write FKolz;
   property Func: String read FFunc write FFunc;
   procedure InputInfo(M: TMemo); override;
   Procedure MLagrang;
   Procedure MNyton;
end;
```

Розглянемо характеристики цього класу.

За полем `MV: DataArr` визначаються таблично задана функція, а за полем `MT: MasTab` визначаються проміжні обчислення для таблично заданої функції.

Так само, як і в класі `TMetSlr`, введено поле `Name: String` для задання назви відображення створюваному об’єкту.

За властивістю

```
property Kolz: Integer read FKolz write FKolz;
```

визначається кількість елементів, що задають таблично задану функцію.

Для збереження отриманого інтерполяційного многочленну введено властивість

```
property Func: String read FFunc write FFunc;
```

Серед методів побудови інтерполяційних многочлена визначені: `MLagrang` – побудова многочленна Лагранжа, `MNyton` – побудова многочленна Ньютона.

Завдання 5 (ППЗ Numet).

Обов’язковий рівень. Розробити клас `TMetAprox`, розмістивши його в модулі `NGAprox`.

```
TMetAprox=class(TNumerMet)
  private
   . . . { опис полів та методів згідно властивостей }
   MT: DataMas;
   function GetTypeName: String; override;
   procedure StartUmovu;
  public
   MV: DataArr;
   property Name: String read FName write FName;
   property Xpr: Extended read FXpr write FXpr;
   property Kolz: Integer read FKolz write FKolz;
   property Func: String read FFunc write FFunc;
   procedure InputInfo(M: TMemo); override;
   . . . { обрані чисельні методи відповідно до класу }
end;
```

Доповнити програму, створену в попередньому завданні, додавши до неї послуги створення екземплярів класу `TMetAprox` та розміщення їх в списку. Передбачити можливість побудови інтерполяційного многочленна Лагранжа.

Підвищений рівень. Виконати завдання обов'язкового рівня та доповнити клас `TMetAprox` методом `Save` за рахунок якого має відбуватися збереження властивостей об'єкту в текстовий файл спеціального виду (ini-файл). Доповнити метод `Load` класу `TNumerMet` операторами, за рахунок яких має відбуватися завантаження екземплярів класу `TMetAprox` з наперед збереженого файлу спеціального виду (ini-файл) до списку об'єктів. Передбачити за рахунок класу `TMetAprox` можливість побудови інтерполяційного многочленна Ньютона.

Клас для генерування HTML документу при виведенні результатів обчислень.

Як зазначалося раніше, при розгляді методу `SInpH` класу `TNumerMet`, виведення результатів може проводитися е вигляді форматованого HTML документу. Для реалізації такої послуги передбачено клас `TGeHtm`, опис якого має вигляд:

```
TGenHtm=class(TObject)
 public
   HtmlList: TStringList;
   Constructor Create;
   Destructor Destroy; override;
   Procedure OutputWindow(WebOut : TwebBrowser);
   Procedure OutputList(StList : String);
   Function FuncToHtm(StFu : String) : String;
   Function GenSysHtm(np, nmax : Integer) : String;
   Procedure GenTabHtm(StGH : String);
   procedure SetFont(Fs : String);
 end;
```

Даний клас містить єдине поле `HtmlList: TStringList`, яке використовується для збереження форматованого HTML документу. Оскільки поле `HtmlList` є агрегатом, то воно має бути створене (використовуючи конструктор) в конструкторі класу `TGenHtm` та знищене в деструкторі даного класу.

Виведення сформованого HTML документу має відбуватися в відповідний компонент середовища розробки, який підтримує HTML форматування, таким

компонентом є `TwebBrowser`. Для забезпечення такого виведення в класі передбачено метод `Procedure OutputWindow(WebOut : TwebBrowser)`.

Додавання нового рядка до поля `HtmlList` відбувається за рахунок методу `Procedure OutputList(StList : String)`. Окрім додавання нового рядка, даний метод містить аналіз спеціальних математичних символів (±, ≈, ≠, ≤, ≥), які передаються в вигляді метасимволів.

Також в класі `TGenHtm` визначені методи, за рахунок яких відбувається генерування HTML розмітки для певних математичних структур

Так за рахунок методу

```
Function FuncToHtm(StFu: String): String;
```

відбувається генерування HTML розмітки для математичних виразів заданих в текстовому вигляді (наприклад: заміна знака множення «*» на «·», запис піднесення до степеня через верхній індекс та ін.).

За рахунок методу

```
Function GenSysHtm(np, nmax: Integer) : String;
```

відбувається генерування, в формат HTML, частини знаку системи в залежності від номеру рівняння та загальної кількості рівнянь в системі («⎧», «⎩», «{»,).

За рахунок методу

```
Procedure TGeHtm.GenTabHtm(StGH: String);
```

відбувається генерування в формат HTML таблиці заданої в вигляді рядка, що передається параметром `StGH`, з використанням певних метасимволів. Кожна таблиця може починатися з символів «\n\» (тут і далі символи « та » використовуються лише для виокрмемлення метасимволів), що вказують на необхідність не відображувати межі таблиці. Далі мають йти значення окремих комірок, порядково, що закінчуються наступним записом метасимволів «\n"k"xyLTRB|\», де: nk – обов’язкові параметри, що задають кількість об’єднаних рядків та стовпців (символи «"» додаються, якщо число n чи k складається з декількох цифр; x – обов’язковий параметр вказує на вирівнювання даних в комірці (може приймати значення: l – по лівому краю, c – по центру, r – по правому краю); y – необов’язковий параметр, який вказує на колір зафарбування

комірки (може приймати значення: q – світло червоний, w – салатовий, e – сірий, за відсутності без зафарбування); LTRB – необов’язкові параметри вони визначають які з границь комірки мають не відображуватися (L – ліва, T – верхня, R – права, B – нижня); символ «|»– необов’язковий параметр, який вказує на те що закінчено рядок.

Так наприклад рядок «\n\C₀ = \21cLTRB\3 - 3\11cLRT\ = 0\21cLTRB|\2\11cLRB|\» матиме вигляд:

$$C_0 = \frac{3 - 3}{2} = 0$$

Розберемо запис рядка:

- присутній метасимвол «\n\», що означає заборону відображення меж таблиці;
- «C₀ =» – вміст комірки;
- «\21cLTRB\» – вказівка для формату комірки: дана комірка буде об’єднувати 2 рядки, об’єднані стовпці відсутні, тексту в комірці буде вирівняний по центру, присутні всі символи «LTRB» тобто жодна з меж комірки не відображається;
- «3 - 3» – вміст комірки;
- «\11cLRT\» – вказівка для формату комірки: дана комірка не буде об’єднувати стовпці та рядки, тексту в комірці буде вирівняний по центру, відсутній символ «B» тобто в комірці буде відображена нижня межа;
- «=0» - вміст комірки;
- «\21cLTRB|\» – вказівка для формату комірки: дана комірка буде об’єднувати 2 рядки, об’єднані стовпці відсутні, тексту в комірці буде вирівняний по центру, присутні всі символи «LTRB» тобто жодна з меж комірки не відображається, завершено введення першого рядка;
- «2» – вміст комірки;
- «\11cLRB|\» – вказівка для формату комірки: дана комірка не буде об’єднувати стовпці та рядки, тексту в комірці буде вирівняний по центру, відсутній символ «T» тобто в комірці буде відображена верхня межа.

За рахунок методу `procedure SetFont(Fs: String)` – відбувається задання шрифту створюваного документу.

Завдання 6 (ППЗ Numet).

Обов'язковий рівень. Розробити клас `TGenHtml`, розмістивши його в модулі `GenHtml`.

```
  TGenHtm=class(TObject)
   private
     HtmlList: TStringList;
   public
     Constructor Create;
     Destructor Destroy; override;
     Procedure OutputWindow(WebOut: TwebBrowser);
     Procedure OutputList(StList: String);
     Function FuncToHtm(StFu: String): String;
     Function GenSysHtm(np, nmax: Integer) : String;
     Procedure GenTabHtm(StGH: String);
     procedure SetFont(Fs: String);
   end;
```

Доповнити програму, створену в попередньому завданні замінивши в ній компоненту, в яку відбувається вивід обчислень, на компоненту `TwebBrowser`. Внести в методи класів зміни, які б дозволяли виводити обчислення в форматі HTML.

Підвищений рівень. Виконати завдання обов'язкового рівня та передбачити метод за рахунок якого буде відбуватися генерування дробових виразів.

Висновки до розділу 2

1. З метою забезпечення у майбутніх учителів математики та інформатики необхідного стартового рівня компетентностей, що необхідні для створення ППЗ, у ВНЗ необхідно здійснювати пропедевтику такого навчання, основою якої є навчання програмуванню та моделюванню. Особливо актуальними є думки щодо такої пропедевтики стосовно вивчення об'єктно-орієнтованого програмування, яке є досить зручним інструментом для створення великих програм. Оскільки лише в процесі

написання достатньо великих програм студент може по-справжньому навчитися аналізувати необхідну предметну галузь, створювати моделі об’єктів предметної галузі, встановлювати зв’язки між ними, будувати відповідне дерево класів, програмувати методи цих класів та поєднувати розроблені класи у різних програмах.

2. У сучасних умовах важливо усвідомити і прийняти принципову педагогічну установку – кожен студент може добровільно обрати для себе рівень засвоєння результатів своєї навчальної діяльності. Для цього досить важливу увагу при навчанні студентів створювати ППЗ необхідно зосередити на рівневу диференціацію навчання, основною особливістю якої є диференціація вимог до набутих компетентностей студентом. Так студентам висувається обов’язковий рівень підготовки, який задає нижню межу засвоєння матеріалу та пропонується обрати дещо вищий рівень засвоєння матеріалу, що відповідає їх потребам, інтересам, здібностям.
3. Застосування саме рівневої диференціації навчання надає можливість викладачу перейти від традиційного способу оцінювання "вирахуванням" до більш зваженого способу оцінювання "складанням", в основу якого покладено необхідність виконання студентом обов’язкового рівня та збільшенням оцінки враховуючи досягнення понад обов’язковий рівень.
4. Для формування програмістських компетентностей майбутніх учителів математики та інформатики необхідно удосконалювати традиційні методи навчання, з огляду на необхідність, у майбутній професійній діяльності вчителя, створювати ППЗ.
5. Для успішного створення ППЗ для певної предметної галузі, розробник має не лише вміти програмувати та бути обізнаним в принципах і підходах створення ППЗ, а і розумітися на термінах, поняттях, правилах відповідної предметної галузі та вміти реалізувати ті чи інші особливості засобами мови та середовища програмування.

6. У процесі навчання створення ППЗ необхідно змістити акцент з використання досить спрощених прикладів, якими переповнені підручники з програмування на аналіз та використання вихідного коду реально використовуваних у педагогічному процесі ППЗ.

Основні результати другого розділу опубліковані в роботах [32; 53; 55; 121; 124; 128; 131; 132; 180; 181]

РОЗДІЛ 3
АНАЛІЗ ЕФЕКТИВНОСТІ РОЗРОБЛЕНИХ КОМПОНЕНТІВ МЕТОДИЧНОЇ СИСТЕМИ

У третьому розділі теоретично обґрунтовано та описано проведення педагогічного експерименту; розглянуті питання перевірки ефективності впровадження компонентів запропонованої методичної системи; з'ясовується необхідність застосування спеціальних методик такого дослідження.

Результати описаного експерименту склали основу висновків щодо справедливості гіпотези, покладеної в основу нашого дослідження.

Педагогічний експеримент, як метод дослідження, полягає в спеціальній організації педагогічної діяльності вчителів і учнів з метою перевірки і обґрунтування наперед розроблених теоретичних припущень або гіпотез. Коли гіпотеза чи припущення знаходить своє підтвердження на практиці, дослідник робить відповідні теоретичні узагальнення і висновки [238, с. 28]. За допомогою педагогічного експерименту встановлюється характер зв'язків між різними компонентами педагогічного процесу, між умовами, факторами та результатами педагогічних дій, перевіряється їх ефективність. Порівнюється ефективність різних факторів або змін у структурі процесу та обирається найкраще для даних умов їх поєднання, виявляються особливості перебігу процесу у нових умовах тощо. При цьому результати експерименту надають можливість встановити закономірні зв'язки між явищами як у якісній, так і в кількісній формах [7].

Педагогічний експеримент повинен перевірити відповідність отриманих результатів гіпотезі, що лежать в основі даного дослідження. А саме: **Гіпотеза дослідження** ґрунтується на припущенні, що цілеспрямоване використання науково обґрунтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів, що базується на використанні вихідних кодів реально використовуваних ППЗ, буде сприяти: активізації пізнавальної діяльності студентів; набуттю практичних навичок створення реальних ППЗ; опануванню особливостей аналізу і

опрацювання математичних об’єктів; покращенню міжпредметних зв’язків, змістовної єдність програмування з іншими навчальними предметами, для яких будуть створюватися ППЗ.

З метою експериментальної перевірки результатів дисертаційного дослідження було проведено педагогічний експеримент в 2006–2013 рр на базі Чернігівського національного педагогічного університету імені Т. Г. Шевченка, Національного педагогічному університеті імені М. П. Драгоманова, Уманського державного педагогічного університету імені Павла Тичини, Криворізького педагогічного інституту при Криворізькому національному університеті, Чернігівського обласного інституту післядипломної освіти імені К. Д. Ушинського, Ніжинського державного університету імені Миколи Гоголя.

3.1. Підхід експертного оцінювання для визначення якості підготовки студентів до розроблення ППЗ

Для проведення педагогічного експерименту з перевірки ефективності впровадження пропонованої методичної системи, першочерговим завданням було добір підходу дослідження, на основі якого можна було б перевірити ефективність запропонованих компонентів методичної системи.

Досить часто результати навчання визначаються шляхом перевірки виконання студентами певних контрольних завдань. Проте, застосовуючи даний метод, досить складно об’єктивно оцінити ефективність підготовки до розроблення ППЗ, що вимагає від студента досить широкого кола знань. Отже, для порівняння результатів навчальної діяльності щодо створення ППЗ було обрано інший підхід, в основу якого покладено оцінювання якості розробленого ППЗ.

Необхідно відмітити, що в сучасних теоретичних і практико-орієнтованих дослідженнях існує декілька підходів до проблеми перевірки та оцінювання якості ППЗ. Серед них можна визначити [239]:

- критеріальне оцінювання їх методичної придатності, що базується на використанні критеріїв якості;

- експериментальна перевірка педагогічної доцільності їх використання, що базується на практичній апробації в процесі навчання у продовж певного періоду;
- експертне оцінювання якості, засноване на компетентній думці експертів, що знають дану галузь знань і мають науково-практичний потенціал для прийняття рішень;
- комплексне оцінювання якості, що інтегрує всі чи деякі з зазначених вище підходів.

Вітчизняний і зарубіжний досвід перевірки якості ППЗ переконує в доцільності застосування саме підходу експертного оцінювання якості, який полягає в наступному: формується група експертів; кожен з цих експертів складає своє власне бачення щодо якості ППЗ, яке базується на власному досвіді та виставляє певні бали. Після проведення експертного оцінювання якості усіма експертами відбувається зведення отриманих результатів в один висновок щодо якості ППЗ. Варто відмітити, що кожний експерт, який приймає участь в процесі оцінювання ППЗ і від думки якого залежить рішення, повинен володіти необхідним компетентностями у відповідній предметній галузі.

Разом з тим експертне оцінювання якості ППЗ не дає гарантій від помилок і виникнення суперечностей у думках різних експертів. Тому в нашому дослідженні зниження ризику прийняття помилкового рішення досягається за рахунок використання наперед визначених критеріїв якості ППЗ. Таке використання визначених критеріїв при проведенні експертного оцінювання ППЗ надає можливість формалізувати якісні та кількісні оцінки експертів. Окрім того, для більш об’єктивного проведення експертного оцінювання ППЗ пропонується сформувати дві групи експертів: фахівці-програмісти та фахівці в предметній галузі. До першої групи варто запросити фахівців-програмістів, які розуміються на використовуваній мові та середовищі програмування. До другої групи варто запросити фахівців з предметної галузі, для дослідження явищ якої розроблено ППЗ, наприклад викладачі методичних кафедр, вчителі-предметники, кращі студенти старших курсів, які вже проходили практику в школі.

Експерти першої групи будуть оцінювати ППЗ з боку ефективності використання апаратної складової комп’ютера та мови і середовища програмування; експерти другої групи будуть оцінювати ППЗ з точки зору вчителя, який використовуватиме даний ППЗ в навчальному процесі. Варто зауважити, що одна людина може виступати експертом в обох групах і проводити експертизу з погляду фахівця-програміста та фахівця відповідної галузі освіти.

В умовах великого різноманіття характеристик ППЗ, різного їх змістовного наповнення досить складно скласти універсальну систему критеріїв якості для всіх класів ППЗ. У зв’язку з цим постала задача в доборі найбільш важливих критеріїв. Враховуючи можливість існування ймовірності того, що не всі суттєві фактори будуть включені в розгляд, в нашому дослідженні ми спиралися на принцип Парето: “При аналізі системи суттєвими є лише деякі фактори, причому 20% із них визначать 80% властивостей системи”.

Виходячи з розглянутих вимог до ППЗ (п. 1.4.1) та показників якості ППЗ (п. 1.4.2) нами було сформульовано перелік критеріїв, згідно яких має проводитися експертне оцінювання ППЗ. Варто зауважити, що кожен член відповідної групи має оцінювати ППЗ за встановленим критерієм у межах максимально визначеної кількості балів. В таблиці 3.1 наведено розроблений нами перелік критеріїв оцінювання ППЗ та вказано максимально можливий бал для оцінки кожного критерію членом певної групи. Оцінювання за деякими критеріями може базуватися на перевірці тестових даних сформульованих групою експертів.

З наведеної таблиці можна бачити, що для членів певних груп не передбачено балів для оцінки окремих критеріїв, тобто не передбачена можливість виконувати оцінювання згідно цього критерію. Це пов’язано з тим, що частина критеріїв мають відношення лише до фахівців однієї з груп.

Згідно проведеного експертного оцінювання розробленого ППЗ, кожен студент чи група студентів, які працювали над конкретним ППЗ, може отримати максимум 100 балів. У подальшому ці бали переводяться пропорційно до необхідної кількості, згідно встановленого розподілу балів.

Таблиця 3.1

Критерії оцінювання ППЗ та розподіл балів за ними

Критерії оцінювання ППЗ	Максимальна кількість балів для групи	
	фахівців-програмістів	фахівці в галузі освіти
1	**2**	**3**
Технічна складова:		
стійкість до помилок і некоректних дій користувача в тому числі і деструктивних	2	2
наявність аналізу помилок при введенні вхідних даних	2	2
коректність роботи ППЗ	2	2
точність обчислень	2	2
робота з периферійним обладнанням (при необхідності)	2	2
забезпечення збереження даних, що опрацьовуються, та повторного їх використання	2	2
забезпечення внесення змін у вхідні дані	2	2
забезпечення відміни виконаних дій (при необхідності)	2	2
легкість у розширенні функціональних можливостей	4	
ефективність використання апаратних ресурсів	4	
Педагогічна складова:		
функціональна повнота згідно встановлених вимог		3
забезпечення моделювання досліджуваних об’єктів, явищ, процесів з предметної галузі		3
врахування своєрідності та особливостей навчального предмету		3
відповідність понять та термінів, що використовуються, рівню підготовки користувачів (учнів або студентів)		3
зрозумілість абревіатур, термінів та пояснень, їх близькість до реальної предметної галузі		3
відображення сучасного стану наукових та педагогічних знань		3
підтримка з боку ППЗ організаційних форм і методів навчання		3
Набуття досвіду експериментально-дослідницької діяльності		3
формування досвіду самоконтролю та самокорекції набутих компетентностей		3
очікуваність інтерфейсу		3

Продовження таблиці 3.1

1	2	3
Ергономічна складова:		
наявність зручного меню та панелей інструментів		3
наявність можливості у користувача використовувати комбінації клавіш для швидкого доступу до послуг ППЗ		3
наявність допоміжних вікон		3
наявність контекстних підказок		3
зручність та простота роботи з ППЗ		3
використання характеристик мови та середовища програмування для забезпечення критеріїв ергономічного рівня	4	
Естетична складова:		
чіткість зображень, побудов, текстових даних		2
відповідність зображень на екрані реальним досліджуваним об’єктам		2
забезпечення налагодження подання даних на екрані		3
використання характеристик мови та середовища програмування для забезпечення критеріїв естетичного рівня	4	

3.2. Практична реалізація та аналіз результатів експериментального дослідження

Відповідно до теорії проведення науково – педагогічних досліджень педагогічний експеримент складається з трьох етапів:

- *констатувального*, задачею якого є збирання та аналіз необхідних даних для виявлення наукової проблеми і висування гіпотези дослідження;
- *пошукового*, задачею якого є збирання і аналіз необхідних даних для уточнення гіпотези та побудови теоретичної моделі дослідження;
- *формувального*, на якому відбувається емпірична перевірка побудованої теоретичної моделі, обґрунтовується вірогідність результатів всього експерименту та здійснюється впровадження основних результатів наукової роботи в практику.

Дослідження проводилося протягом 2006-2013 років.

Метою *констатувального етапу* педагогічного експерименту (2006-2007 рр.) було:

- вивчення теоретичного стану дослідженості розглядуваної проблеми шляхом аналізу психолого-педагогічної, наукової та навчально-методичної літератури;
- аналіз проектів стандартів вищої освіти, навчальних планів і навчальних програм з технологій створення ППЗ, а також навчальних планів і програм з дисциплін, в межах яких може відбуватися таке навчання;
- аналіз існуючих навчальних посібників, як українських так і закордонних, які можуть бути використані в процесі підготовки до створення ППЗ в вищій школі;
- встановлення критеріїв оцінювання рівня компетентностей з програмування та ООП майбутніх учителів математики та інформатики;
- вивчення вітчизняного і зарубіжного досвіду навчання програмування та створення ППЗ;
- аналіз змісту лабораторних робіт з програмування та ООП та їх ролі в системі підготовки студентів до розроблення ППЗ.

У ході констатувального етапу педагогічного експерименту застосовувалися пасивні методи дослідження поведінки студентів у процесі навчання, що тільки реєструють їх стан протягом традиційного навчання, зокрема аналіз результатів виконання лабораторних робіт та результати успішності виконання поточних та підсумкових контрольних робіт. Студенти, які брали участь в експерименті, вивчали курс з програмування згідно навчальної програми, яка відповідала вимогам тимчасових стандартів вищої освіти. Ніяких додаткових завдань, крім тих, що було заплановано навчальною програмою, студентам не ставилося. Крім того, для отримання деяких даних було обрано метод анкетування, який забезпечив отримання даних, які надали можливість адекватно і обґрунтовано описати стан проблеми, що досліджується (приклади анкет для студентів та вчителів подану в додатку Ж).

Аналіз результатів проведення констатувального етапу педагогічного експерименту надав можливість виявити недостатню методичну розробленість питань створення ППЗ, необхідність цілеспрямованого формування знань щодо

розробки ППЗ у майбутніх учителів математики та інформатики та дав можливість з'ясувати основні напрямки формування компонентів методичної системи підготовки майбутніх учителів математики і інформатики до розробки ППЗ.

Дані висновки знаходили багаторазове підтвердження в бесідах з учителями інформатики, що працюють у навчальних закладах різних регіонів України, а також з колегами під час міжнародних і всеукраїнських наукових та науково-методичних конференцій і семінарів.

Вірогідність результатів констатуючого етапу педагогічного експерименту визначається повнотою набору проаналізованих літературних джерел та програм з програмування та ООП.

На *пошуковому етапі* педагогічного експерименту (2007-2009 рр.) відповідно до мети дослідження були розв'язані такі завдання:

- теоретично обґрунтовано та уточнено ключові положення концепції створення компонентів методичної системи;
- визначено зміст теоретичної частини та лабораторних робіт дисципліни "Технології програмування та створення ППЗ";
- визначено методи, засоби та форми організації навчання, які в подальшому були покладені в основу компонент пропонованої методичної системи навчання програмування і розробки ППЗ;
- проведено пошук і методичний аналіз мов та середовищ програмування для їх подальшого використання в навчальному процесі;
- проведено пошук способів організації навчальної діяльності, спрямованих на підвищення практичної значущості результатів навчання;
- визначено дисципліни, в межах яких доцільне проведення пропедевтичних заходів для забезпечення подальшого педагогічного ефекту під час навчання створювати ППЗ.

Результатами пошукового етапу педагогічного експерименту можна вважати наступне:

- уточнена гіпотеза дисертаційного дослідження;

- розроблено окремі компоненти методичної системи підготовки майбутніх учителів математики та інформатики до розробки ППЗ, які викладені в навчально-методичному посібнику „Теорія і методика розробки педагогічних програмних засобів” [53];
- визначено і обґрунтовано вибір ПЗ, що використовуватиметься для оволодіння компетентностями розроблення ППЗ;
- розроблені авторські програми курсів «Об’єктно-орієнтоване програмування» та «Технології програмування та створення ППЗ»;
- вдосконалено ППЗ «Gran2D» [180] та розроблено «Numet» [181].

Окрім того, визначилася опора на вільно поширюване програмне забезпечення для вивчення основ алгоритмізації, основ об’єктно-орієнтованого програмування – вільно поширювані середовища програмування FreePascal та Lazarus. Середовище Lazarus використовувалося також при вивченні дисципліни “Технології програмування та створення ППЗ”.

На *формувальному етапі* педагогічного експерименту (2009-2013 рр.) брали участь 426 студентів, з них у контрольній групі було 218 студентів, у експериментальній групі – 208 студентів.

Мета проведення педагогічного експерименту на формувальному етапі полягала в перевірці на практиці ефективності розроблених компонентів комп’ютерно орієнтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення ППЗ та вдосконалені її окремих компонент.

Для перевірки результатів навчальної діяльності щодо розроблення ППЗ було обрано підхід, в якому вагоме місце займає експертне оцінювання якості розробленого ППЗ, що спирається на запропонований перелік критеріїв якості ППЗ.

Формування контрольної та експериментальної груп здійснювалося на основі існуючих навчальних груп за результатами екзаменаційних оцінок отриманих студентами з курсу “Алгоритмізації та програмування” та “Об’єктно-орієнтованого програмування” та за результатами попереднього тестування

(орієнтовний перелік пропонованих тестових завдань подано в додатку 3), щоб забезпечити статистичну однаковість рівнів компетентностей студентів контрольної та експериментальної груп.

Результати тестів, по ECTS шкалі, з врахуванням яких проводився поділ на контрольну та експериментальні групи подано в таблиці 3.2.

Таблиця 3.2.

Оцінки вхідного тестування

	Кількість балів						
	FX	F	E	D	C	B	A
Контрольна група	7	10	45	55	39	36	26
Експериментальна група	5	11	48	47	41	29	27

Висунемо гіпотезу H_0, про те що експериментальні та контрольні групи студентів є статистично однакові щодо сформованості компетентностей з програмування. Перевірку цієї гіпотези здійснено, використовуючи критерій Пірсона χ^2 за допомогою програми "*GRAN1*", при рівні значущості $\alpha = 0{,}95$ (рис. 3.1).

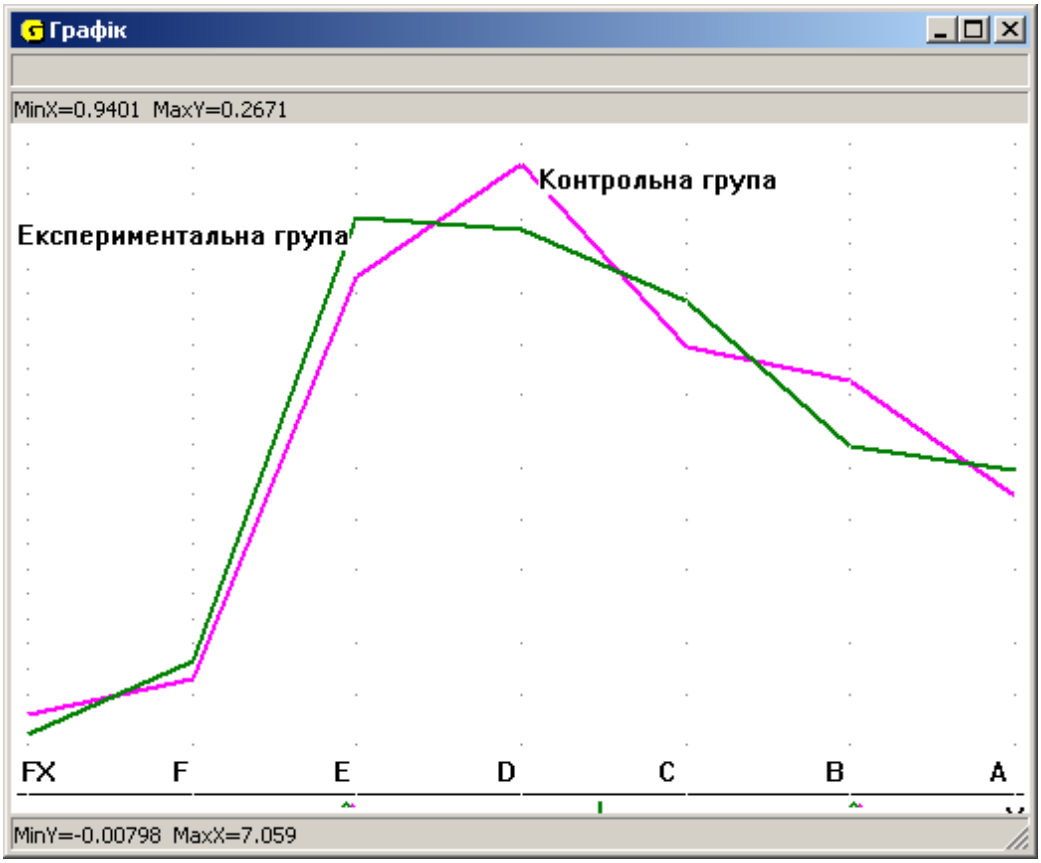


Рис. 3.1. Результати вхідного тестування

При цьому отримано, $\chi^2_{åêñï} = 3{,}61$, $\chi^2_{теор} = 12{,}59$, тобто $\chi^2_{експ} < \chi^2_{теор}$, а отже немає причин вважати гіпотезу H_0 такою, що суперечить статистичним даним. Таким чином, можна стверджувати, що експериментальні та контрольні групи студентів, що брали участь в експерименті, статистично однакові щодо сформованості компетентностей з курсу програмування та ООП.

Заняття з дисципліни “Технології програмування та створення педагогічних програмних засобів” в контрольних групах проводились за методичною системою, обраною викладачем, найчастіше з використанням досить спрощених прикладів створення окремих елементів ППЗ. Заняття в експериментальних групах проводились з використанням методичної системи, розробленої під час пошукового етапу дослідження.

Як уже було зазначено раніше, ефективність компонентів методичної системи навчання в обох групах студентів, в більшій мірі, визначалась за результатами проведеного експертного оцінювання розроблених студентами ППЗ, як результату підготовки до розроблення ППЗ. Бали при оцінюванні якості ППЗ виставлялися експертами згідно визначених показників (п. 3.1). В подальшому отримані бали за розроблені ППЗ були пропорційно переведені в передбачену за дисципліною кількість балів та доповнені балами за підсумковий тест (Орієнтовний перелік пропонованих тестових завдань подано в додатку И). Результати оцінювання за результатами експерименту подано в таблиці 3.3.

Таблиця 3.3.

Оцінки за результатами експерименту

	Кількість балів						
	FX	F	E	D	C	B	A
Контрольна група	4	9	41	56	42	39	27
Експериментальна група	3	8	23	44	55	45	30

Після проведення експерименту вибірки контрольної та експериментальної групи мають статистичні відмінності, оскільки $\chi^2_{åêñï} = 20{,}52$, $\chi^2_{òåîð} = 12{,}59$, тобто $\chi^2_{åêñï} > \chi^2_{òåîð}$ (рис. 3.2).

Це дає підстави стверджувати, що після проведення експериментального навчання контрольні та експериментальні групи студентів за сформованістю компетентностей статистично відрізняються.

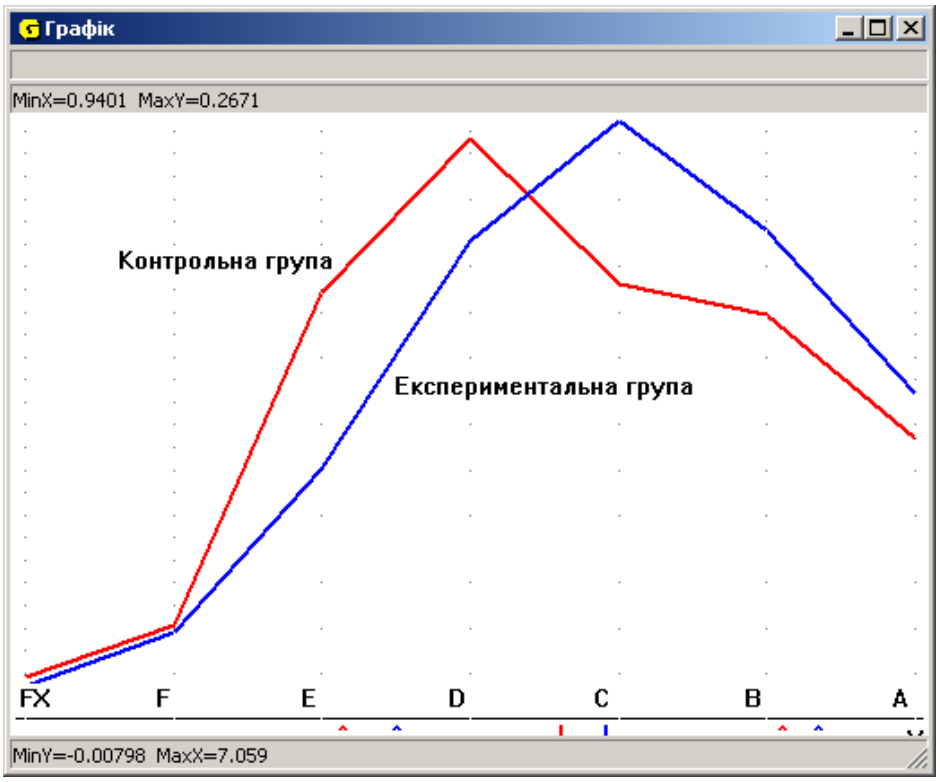


Рис. 3.2. Результати експерименту

Враховуючи, що в експериментальній групі кількість оцінок «A», «B» та «C» зросла за рахунок зменшення кількості оцінок «D», «E», «F» та «FX», можна зробити висновок, що використання пропонованої методичної системи з використанням вихідних кодів реально використовуваних в педагогічному процесі ППЗ в експериментальній групі призвело до підвищення рівня підготовки до розроблення ППЗ.

На основі аналізу отриманих результатів можна зробити висновок про те, що цілком підтверджується гіпотеза про те, що впровадження в навчальний процес розроблених компонентів методичної системи надає можливість інтенсифікувати процес навчання, активізувати пізнавальну діяльність студентів, підняти межу базового рівня знань студентів, диференціювати й індивідуалізувати процес навчання з огляду на різний рівень здібностей студентів, підвищити рівень підготовленості студентів до створення ППЗ, підвищити рівень їх професійної підготовки.

Вірогідність одержаних висновків визначається тривалістю експерименту; кількістю учасників експерименту; репрезентативністю вибірки; інтерпретацією результатів, що отримані, за допомогою методів математичної статистики.

Висновки до розділу 3

Педагогічний експеримент повинен був підтвердити відповідність отриманих результатів гіпотезі, що лежить в основі даного дослідження.

У результаті дослідження детально вивчено проблему перевірки ефективності впровадження пропонованих компонентів комп'ютерно орієнтованої методичної системи. З'ясовано необхідність застосування особливих методик такого дослідження, що виключають вплив особливостей мови та використовуваних парадигм програмування на результати дослідження.

Адаптовано один з підходів оцінки якості ППЗ, а саме проведення експертного оцінювання якості ППЗ групою експертів з використанням запропонованого переліку критеріїв. З використанням адаптованого підходу проведено порівняння ефективності підготовки до розроблення ППЗ за рахунок використання запропонованих компонентів методичної системи.

Результати експериментальної перевірки ефективності впровадження запропонованих компонентів методичної системи підготовки майбутніх учителів математики та інформатики до розроблення ППЗ вказують на доцільність використання вихідних кодів реально використовуваних ППЗ при такій підготовці.

Запропонований варіант використання експертного оцінювання для перевірки ефективності впровадження розроблених компонентів запропонованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення ППЗ має окремі недоліки: складність у формуванні груп експертів, що володіють необхідним компетентностями у відповідній предметній галузі чи мають досвід; можливе виникнення суперечностей у думках різних експертів; недостатня кількість показників якості для експертизи окремих типів ППЗ.

Отже, впровадження розроблених компонентів методичної системи відкриває можливості для ефективного подолання раніше зазначених труднощів у реалізації навчання розроблення ППЗ; надає можливість інтенсифікувати та індивідуалізувати процес навчання за рахунок рівневої диференціації; надає можливість раціонально збалансувати обсяг теоретичного матеріалу і практичної роботи, поєднати високий науковий рівень навчального матеріалу з доступністю його для студентів; зумовлює посилення регулярності навчальної діяльності студентів та підвищення їхньої самостійної роботи, що, поряд із іншим, надає можливість значно підвищити рівень професійної підготовки майбутніх викладачів.

Основні результати третього розділу опубліковані в роботах [53; 74; 120; 129].

ВИСНОВКИ

У даній дисертаційній роботі розглянута можливість підвищення професійного рівня майбутніх учителів математики та інформатики що до розроблення педагогічних програмних засобів за рахунок цілеспрямованого використання вихідного коду реально використовуваних в педагогічному процесі ППЗ.

У процесі проведення дисертаційного дослідження було вирішено усі поставлені завдання і відповідно до мети та висунутої гіпотези отримано такі **результати**:

1. Проаналізовані питання навчання програмування у педагогічних ВНЗ, узагальнено досвід викладачів та з'ясовані особливості організації та здійснення навчання програмування в цілому та навчання програмування у педагогічних ВНЗ.

2. Проведено аналіз, систематизацію, узагальнення наукової вітчизняної та зарубіжної фахової, педагогічної та навчально-методичної літератури; з'ясовано стан дослідженості проблеми щодо навчання майбутніх учителів проектування та створення ППЗ. На основі отриманих результатів визначено умови та шляхи реалізації процесу підготовки майбутніх учителів математики та інформатики до розроблення ППЗ; визначено концептуальні засади, підходи до створення ППЗ та етапи проектування і створення ППК.

3. Визначено і обґрунтовано вибір мови програмування Free Pascal та середовища програмування Lazarus, використання яких має забезпечувати оволодіння компетентностями щодо розроблення ППЗ.

4. Визначено і розроблено основні компоненти комп'ютерно орієнтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення ППЗ, яка базується на використанні вихідних кодів ППЗ Numet та ППЗ Gran2d, що були розроблені за безпосередньої участі дослідника. Розроблено практичні завдання за двома рівнями для забезпечення рівневої диференціації навчання та перелік тестових питань для перевірки знань студентів.

5. Для перевірки результатів навчальної діяльності щодо розроблення ППЗ було обрано підхід, в якому вагоме місце займає експертне оцінювання якості розробленого ППЗ на основі визначених критеріїв, та сформульовано перелік таких критеріїв.

6. У ході педагогічного експерименту з'ясовано ефективність розроблених компонентів комп'ютерно орієнтованої методичної системи підготовки підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів.

Аналіз отриманих результатів дисертаційного дослідження дає можливість зробити такі **висновки**:

1. Рівень сформованості компетентностей майбутніх учителів математики та інформатики Українських ВНЗ щодо розробки ППЗ не відповідає повною мірою вимогам сьогодення. Однією із головних причин такого стану є складність, а іноді і неможливість, ґрунтовної теоретичної та практичної підготовки до розроблення ППЗ в умовах обмеженої кількості годин та традиційної організації навчального процесу. Це дало підстави зробити висновок про те, що проблема підготовки майбутніх учителів математики та інформатики до розроблення ППЗ залишається недостатньо розробленою.

2. У процесі пошуку ефективних шляхів удосконалення навчання програмування та створення ППЗ, на основі вивчення психолого-педагогічної, науково-методичної та навчальної літератури, здобутків передового педагогічного досвіду, а також аналізу власної практикої роботи у педагогічному ВНЗ, встановленно, що одним із ефективних чинників підвищення якості підготовки до розроблення ППЗ є застосування вихідних кодів реально впроваджених і широко використовуваних ППЗ. Запровадження комп'ютерно орієнтованої методичної системи навчання розроблення ППЗ, побудованої на цій основі, надає можливість ефективно формувати належні компетентності у студентів із різним рівнем здібностей.

3. На основі аналізу підходів, що застосовуються до навчання програмування, було визначено особливості організації підготовки до

розроблення ППЗ та необхідність в поєднанні навчально-методичних підходів для підвищення рівня набуття відповідних компетентностей.

4. Аналіз сучасних тенденцій в галузі розробки педагогічних програмних засобів дав можливість визначити пріоритетність об’єктно-орієнтованої парадигми програмування і відповідно об’єктно-орієнтовних мов програмування для розроблення ППЗ, оскільки безперечною перевагою об’єктно-орієнтованої парадигми є концептуальна близькість до предметної галузі довільної структури та призначення.

5. Апробація під час дослідно-експериментальної роботи комп’ютерно орієнтованої методичної системи підготовки до розроблення ППЗ майбутніх учителів математики та інформатики на основі використання вихідних кодів реально використовуваних в педагогічній практиці ППЗ (Numet та Gran2d) дала можливість визначити умови, необхідні для підвищення ефективності підготовки до розроблення ППЗ, та обґрунтовано стверджувати, що така модель являє собою структурно-функціональну конструкцію, в якій фігурують взаємопов’язані та взаємозумовлені між собою компоненти. За даними компонентами визначається сутність процесу підготовки до розроблення ППЗ як результат взаємозалежності професійної й особистісної характеристик студента та фундаментальної підготовки викладача.

6. У результаті виконаного дослідження було підтвержено висунуту гіпотезу про ефективність цілеспрямованого використання науково обґрунтованої методичної системи підготовки майбутніх учителів математики та інформатики до розроблення педагогічних програмних засобів, що базується на використанні вихідних кодів реально використовуваних ППЗ, та з’ясовано, що впровадження такої методичної системи надає можливість суттєво активізувати пізнавальну діяльність студентів, набуття практичних навичок створення реальних ППЗ, опанувати особливості аналізу і опрацювання математичних об’єктів; покращити міжпредметні зв’язки, змістовну єдність програмування з іншими навчальними предметами, для яких будуть створюватися ППЗ.

7. На основі аналізу результатів виконаного дослідження сформульовані рекомендації щодо підготовки майбутніх учителів математики та інформатики до розроблення ППЗ. Підтверджено доцільність впровадження розроблених автором дисертаційного дослідження методичних рекомендацій щодо пропедевтичного навчання студентів розробляти ППЗ.

Проведене дослідження не вичерпує всіх аспектів проблеми підготовки до розроблення ППЗ майбутніх учителів математики та інформатики. До напрямів, що потребують подальшого дослідження, можна віднести: можливість конкретизації та доповнення змісту пропонованих компонентів методичної системи, вивчення механізму впливу навчання розроблення ППЗ на особистісну самореалізацію студента у його майбутній професійній діяльності, можливе розширення навчального матеріалу за рахунок використання вихідних кодів інших ППЗ, дослідження можливості застосування окремих компонентів розробленої методичної системи до створення ППЗ для підтримки навчання не лише математичних дисциплін, визначення шляхів вдосконалення існуючих педагогічних програмних засобів та створення нових ППЗ, зокрема web-орієнтованих з врахуванням психологічних, дидактичних та методичних особливостей навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. А. с. № 7937. Програмний комплекс GRAN : комп’ютерна програма / М. І. Жалдак, О. В. Вітюк, Ю. В. Горошко. – 11.07.2003.
2. Абдувахидов С. Н. О формировании готовности студентов университетов к педагогической деятельности / С. Н. Абдувахидов // Теория и практика высшего педагогического образования: Межвузовск. сб. науч. тр. – М., 1987. – С. 131-134.
3. Агафонов В. Н. Спецификация программ: понятийные средства и их организация / В. Н. Агафонов. – Новосибирск: Наука (Сибирское отделение), 1987. – 350 с.
4. Ананьев Б. Г. Человек как предмет познания / Б. Г. Ананьев. – Л.: Издательствово ЛГУ, 1969. – 339 с.
5. Антропова Н. . Дифференцированное обучение: педагогическая и философская оценка / Н. В. Антропова, Г. Г. Манке // Педагогика. – 1992. – № 9. – С. 23-28.
6. Ахо А. В. Структуры данных и алгоритмы / А. В. Ахо , Д. Э. Хопкрофт, Д. Д. Ульман; [пер. с англ.]. – М. : Издательский дом «Вильямс», 2001. – 384 с.
7. Бабанский Ю. К. Проблемы повышения эффективности педагогических исследований / Юрий Константинович Бабанский. – М. : Педагогика, 1982. – 192 с.
8. Бабанский Ю. К. Дидактические проблемы совершенствования учебных комплексов / Ю. К. Бабанский // Проблемы школьного учебника. – М.: Просвещение, 1980. – Вып. 8. – С. 17–33.
9. Бабанский Ю. К. Методы обучения в современной общеобразовательной школе / Ю. К. Бабанский. – М.: Просвещение, 1985. – 186 с.
10. Бабанский Ю. К. Оптимизация процесса обучения: Общедидактический аспект / Ю. К. Бабанский. – М.: Педагогика, 1977. – 96 с.

11. Бакуменко К. В. Проведення обчислювального експерименту засобами системи дистанційного вивчення курсу «Основи алгоритмізації та програмування» / О. В. Співаковський, Н. В. Осипова, К. В. Бакуменко // Інформаційні технології в освіті: зб.наукових праць. – Херсон: ХДУ,2010. – № 6. – С. 1-22.
12. Балл Г. А. Теория учебных задач: Психолого-педагогический аспект / Георгий Алексеевич Балл. – М.: Педагогика, 1990. – 184 с.
13. Балл Г. О. Про психологічні засади формування готовності до професійної праці / Г. О. Балл // Психолого-педагогічні проблеми професійної освіти: науково методичний збірник – К., 1994. С. 98-100.
14. Балл Г. А. Категория задачи и ее значение для психолого-педагогических исследований / Г. А. Балл, Г. С. Костюк // Вопросы психологии – 1977. – №3. – С. 12-23.
15. Бежанова М. М. Современные понятия и методы программирования / М. М. Бежанова, И. В. Поттосин. – М.: Научный мир, 2000. – 192с.
16. Бережинська Т. В. Готовність вчителя до оцінювання навчальних досягнень молодших школярів / Т. В. Бережинська // Психолого-педагогічні проблеми сільської школи: збірник наукових праць. – Умань: УДПУ ім. Павла Тичини, 2002. – № 2. – С. 134-138.
17. Биков В. Ю. Моделі організаційних систем відкритої освіти: Монографія / В. Ю. Биков. – К.: Атіка, 2008. – 684 с: іл.
18. Биков В. Ю. Навчальна програма з інформатики для 8-11 класів загальноосвітніх навчальних закладів універсального та фізико-математичного профілю./ В. Ю. Биков, В. Д. Руденко // Комп’ютер у школі та сім’ї – 2005, №1. С. 17–33.
19. Бим-Бад Б. М. Педагогический энциклопедический словар / Б. М. Бим-Бад. – М., 2002. – 528 с.
20. Блонский П. П. Память и мышление / П. П. Блонский. – СПб.: Питер, 2001. – 357 с.

21. Богданова Л. Д. Выбор средств педагогического влияния на формирование информационной культуры студентов на основе межпредметных связей: результаты эксперимента. / Л. Д. Богданова // Проблеми інженерно-педагогічної освіти. – 2006. – № 12. – С. 179-188.
22. Борисов В. М. Разработка пакетов программ вычислительного типа. / В. М. Борисов. – М.: Издательство МГУ, 1990. – 123 с.
23. Бороненко Т. А. Теоретическая модель системы методической подготовки учителя информатики : дис. ... доктора пед. наук : 13.00.02 / Т. А. Бороненко. – СПб., 1997. – 335 с.
24. Братко И. Программирование на языке Пролог для искусственного интеллекта / И. Братко; [пер. с англ.]. – М.: Мир, 1990. – 560 с., ил.
25. Брызгалова С. И. Формирование в вузе готовности учителя к педагогическому исследованию: теория и практика / С. И. Брызгалова. – Калининград, 2004. – 312 с.
26. Бурда М. І. Рівнева диференціація у шкільній математиці: Методика. Досвід / М. І. Бурда, В. В. Дивак, П. М. Литвиненко // Рідна школа. – 1994. – № 8. – с. 56-60.
27. Буч Г. Объектно-ориентированное проектирование с примерами применения / Г. Буч; [пер. с англ.]. – К.: Диалектика; М.: АО «И.В.К», 1992. – 230 с.
28. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Гради Буч; [пер. с англ.]. – М.: Издательство БИНОМ, СПб.: Невский диалект, 1999. – 560 с.
29. Вендеров А. М. Проектирование программного обеспечения экономических информационных систем / А. М. Вендеров. – М.: Финансы и статистика, 2002. – 348 с.
30. Вендров А. М. CASE-технологии. Современные методы и средства проектирования информационных систем [Электронный ресурс] / А. М. Вендров // Веб-сайт «case-tech.h1.ru». – Режим доступу: http://case-tech.h1.ru/library/vendrov/index.htm. – Назва з екрана.

31. Вернигоренко С. А. Вивчення основ об’єктно-орієнтованого програмування у класах фізико-математичного профілю / С. А. Вернигоренко // Комп’ютерно-орієнтовані системи навчання: збірник наукових праць. М-во освіти і науки України, НПУ ім. М. П. Драгоманова; Відп. ред. М. І. Жалдак. – Київ, 2011. – Вип. 12 – С.61 – 74.
32. Вінниченко Є. Ф. Деякі особливості геометричних перетворень в програмі GRAN-2D / Є. Ф. Вінниченко, А. О. Костюченко // Науковий часопис Національного педагогічного університету імені М. П. Драгоманова. Серія 2: Комп’ютерно-орієнтовані системи навчання. – К.: НПУ імені М. П. Драгоманова, 2007. – № 5(12). – С. 114-120.
33. Власенко К. В. Актуалізація евристичних ситуацій на уроках геометрії (за матеріалами основної школи) / К. В. Власенко, О. І. Скафа. – Х. : Вид. група «Основа», 2010. – 159 с.
34. Волинський В. П. Методичні рекомендації до використання педагогічних програмних засобів у навчальному процесі / В. П. Волинський, Г. О. Козлакова. – К.: НПУ ім. М. П. Драгоманова, 2007. – 59 с.
35. Волошинов С. А. Алгоритмічна підготовка судноводіїв в умовах інформаційно-комунікаційного педагогічного середовища / С. А. Волошинов // Інформаційні технології в освіті. – 2010. – № 8. – С. 103-108.
36. Волошинов С. А. Алгоритмічна підготовка майбутніх судноводіїв з системою візуальної підтримки в умовах інформаційно-комунікаційного педагогічного середовища: дис. … кандидата пед. наук: 13.00.04 «теорія і методика професійної освіти » / Сергій Анатолійович Волошинов / Херсонський державний університет, Херсон, 2012. – 245 с.
37. Вострокнутов И. Е. Оценка визуальных сред на экране монитора или почему болят глаза при работе на компьютере / И. Е. Вострокнутов // Информатика и образование. – 2002. – № 1. – С. 64-67.
38. Всемирная энциклопедия: Философия / [Главн. науч. ред. и сост. А. А. Грицанов]. – М.: АСТ, 2001. – 1312 с.

39. Габрусєв В. Ю. Зміст і методика вивчення шкільного курсу інформатики на основі вільно поширюваної операційної системи Linux: дис. … кандидата пед. наук : 13.00.02 / Валерій Юрійович Гарбусєв / НПУ імені М. П. Драгоманова, – К., 2003. – 221 с.
40. Галузеві стандарти вищої освіти. Напрям підготовки 0101 Педагогічна освіта. Спеціальність 6.010100 Педагогіка і методика середньої освіти. Математика. – К.: Вид-во НПУ імені М. П. Драгоманова, 2003. – 148 с.
41. Ганжела С. І. Формування пізнавальної самостійності учнів основної школи в навчанні геометрії з використанням інформаційних технологій: дис. ... канд. педаг. наук : 13.00.02 / С. І. Ганжела ; НПУ ім. М. П. Драгоманова. – К., 2010. – 255 с.
42. Глинський Я. М. Інформатика: 10-11 класи: Навч. посіб.: У 2 ч. – ч. 1.: Алгоритмізація й програмування. / Я. М. Глинський / 7-ме вид. – Львів: СПД Глинський, 2007. – 56 с.
43. Глинський Я. М. Переваги застосування мови програмування JAVA в навчальному процесі / Я. М. Глинський, В. Є. Анохін, В. А. Ряжська. // Інформатика та інформаційні технології в навчальних закладах : Науково-методичний журнал. – К. : Освіта України, – 2005. – № 4 – С.34–38.
44. Гончаренко С. У. Український педагогічний словник/ С. У. Гончаренко. – К.: Либідь, 1997. – 376 с.
45. Гончаров Н. К. Дифференциация и индивидуализация образования в современных условиях / Н. К. Гончаров // Проблема социальной педагогики. – М.: Педагогика, 1973. – № 3. – С. 65.
46. Гончарова І.В. Евристики в геометрії / І.В.Гончарова, О.І.Скафа. – Х. : Вид. група “Основа”, 2004. – 112 с.
47. Горошко Ю. В. Активізація пізнавальної діяльності учнів на уроках математики з використанням НІТ. Проблеми інформатизації освіти / Ю. В. Горошко, А. В. Пеньков. – К.: УДПУ, 1994.- С. 47-54.
48. Горошко Ю. В. Використання комп’ютерних програм для створення динамічних моделей при вивченні математики / Ю. В. Горошко,

Є. Ф. Вінниченко // Науковий часопис НПУ імені М. П. Драгоманова. Серія № 2. Комп'ютерно-орієнтовані системи навчання: зб. наук. праць. – К., 2006. – № 4(11). – С. 56-62.

49. Горошко Ю. В. Метод найменших квадратів та його реалізація засобами НІТ / Ю. В. Горошко // Комп'ютерно-орієнтовані системи навчання: зб. наук. праць. – К.: НПУ імені М. П. Драгоманова, – 2003. – Вип. 6. – С. 106-112.
50. Горошко Ю. В. Методика вивчення ППЗ GRAN-2D на уроках інформатики та його застосування в планіметрії / Ю. В. Горошко, Л. Грамбовська // Комп'ютер у школі та сім'ї. – 2008. – № 3. – С. 14-22.
51. Горошко Ю. В. Обласні олімпіади з інформатики на Чернігівщині / Ю. В. Горошко, А. О. Костюченко // Комп'ютер у школі та сім'ї. – 2007. – № 7(63). – С. 47-50.
52. Горошко Ю. В. Система інформаційного моделювання у підготовці майбутніх учителів математики та інформатики : дис. ... доктора пед. наук : 13.00.02 «теорія та методика навчання (інформатика)» / Юрій Васильович Горошко; НПУ ім. М. П. Драгоманова. – К., 2013. – 470 с.
53. Горошко Ю. В. Теорія і методика розробки педагогічних програмних засобів / Ю. В. Горошко, А. О. Костюченко. – Чернігів: Виготовлення Єрмоленко О. М., 2011. – 144 с.
54. Горошко Ю. В. Використання ВПЗ у вивченні основ програмування / Горошко Ю. В., Костюченко А. О., Шкардибарда М. І. // Інформатика та інформаційні технології. – 2012. – № 1. – С. 22-25.
55. Горошко Ю. В. Використання ППЗ „NUMET" при вивченні елементів чисельних методів / Ю. В. Горошко, А. О. Костюченко // Вісник Чернігівського державного педагогічного університету імені Т. Г. Шевченка. Серія: Педагогічні науки. – 2003. – Вип. 19. – С. 26-30.
56. Горошко Ю. В. Проблеми та особливості впровадження вільного програмного забезпечення у навчальний процесс / Горошко Ю. В., Костюченко А. О., Шкардибарда М. І. // Комп'ютер у школі та сім'ї. – 2010.– № 7. – С. 8-10.

57. Грамбовська Л. В. Особистісно орієнтоване навчання геометрії в основній школі: дис. ... канд. педаг. наук : 13.00.02 / Л. В. Грамбовська; НПУ ім. М. П. Драгоманова. – Київ, 2009. – 313 с.
58. Гришко Л. В. Концептуальні підходи до навчання основ програмування у вищій школі / Л. В. Гришко // Науковий часопис Національного педагогічного університету імені М. П. Драгоманова. Серія 2 : Комп'ютерно-орієнтовані системи навчання. – К.: НПУ імені М. П. Драгоманова, 2004. – № 1(8). – С. 134–147.
59. Гришко Л. В. Методична система навчання основ програмування майбутніх інженерів-програмістів : дис. ... кандидата пед. наук : 13.00.02 «теорія та методика навчання (інформатика)» / Людмила Веніамінівна Гришко; Черкаський національний університет імені Богдана Хмельницького. – К., 209. – 281 с.
60. Гришко Л. В. Навчання стилю програмування, як складова формування професійної культури майбутнього інженера-програміста / Л. В. Гришко // Вісник Черкаського університету, серія «Педагогічні науки». Випуск 143. – Черкаси, 2009. – С. 37-43.
61. Гришко Л. В. Психолого-педагогічні аспекти навчання основ програмування / Л. В. Гришко // Науковий часопис НПУ імені М.П. Драгоманова. Серія №2: Комп'ютерно-орієнтовані системи навчання. – К.: НПУ імені М.П. Драгоманова, 2003. – № 19. – С. 110-115.
62. Гришко Л. В. Пути индивидуализации процесса обучения основам программирования / Л. В. Гришко, Н. В. Чернявский // Комп'ютерне моделювання та інформаційні технології в науці, економіці та освіті. V Всеукраїнська науково-практична конференція (м. Черкаси, 21 – 23 квітня 2003 р.). – Черкаси, 2003. – С.27-29.
63. Гэннон Дж. Принципы разработки программного обеспечения / М. Зелковец, А. Шоу, Дж. Гэннон. – М.: Мир, 1982. – 368 с.
64. Даль В. И. Толковый словарь живого великорусского языка / В. И. Даль. – М.: Цитадель, 1998.

65. Данилов М. А. Дидактика средней школы. Некоторые проблемы современной дидактики / М. А. Данилов, М. Н. Скаткин. – М.: Просвещение, 1975. – 303 с.
66. Дейкстра Э. Дисциплина программирования / Э. Дейкстра; [пер. с англ.]. – М.: Издательство «Мир», 1978. – 274 с.
67. Декларативне програмування [Електронний ресурс] // Вікіпедія – вільна енциклопедія. – Режим доступу: http://uk.wikipedia.org/wiki/Декларативне_програмування (28.06.2013). – Назва з екрану.
68. Державна національна програма «Освіта. Україна XXI століття». – К.: Райдуга, 1994. – 61 с.
69. Державний стандарт освітньої галузі “Технології” : (проект) для загальноосвіт. серед. шк. / В. Ю. Биков, М. І. Жалдак, Н. В. Морзе та ін. // Освіта України. – 2003. – № 3-4. – 10 с.
70. Деркач А. А. Взаимосвязь структурных компонентов состояния психической готовности студентов к педагогической деятельности / А. А. Деркач // Психолого-педагогические проблемы взаимодействия учителя и учащихся: Сб. науч. трудов. – М.,1980. С. 141-149.
71. Джозеф Ф. Программное обеспечение и его разработка / Фокс Джозеф; [перевод с англ. Л. Е. Карпова]; под ред. Д. Б. Подшивалова. – М.: Мир, 1985. – 368 с.
72. Диференціація у навчанні математики / Г. В. Дорофєєв, Л. В. Кузнєцова, С. Б. Суворова, В. В. Фірсов // Математика в школі. – 1990. – № 4. – С.15.
73. Дідковська М. В. Методи оцінки та засоби підвищення надійності програмного забезпечення: дис... канд. техн. наук: 05.13.06 / Марина Віталіївна Дідковська / Національний технічний ун-т України "Київський політехнічний ін-т". – К., 2006. – 181 с.
74. Дубина К. М. Комп’ютерне тестування, як засіб контролю знань / К. М. Дубина, А. О. Костюченко // Вісник Чернігівського державного

педагогічного університету імені Т. Г. Шевченка. Серія: Педагогічні науки. – 2008. – Вип. 58. – С. 65-70.

75. Дурай-Новакова К. М. Формирование профессиональной готовности студентов к педагогической деятельности: автореф. дис. на здобуття наук. ступеня канд. пед. наук: спец 13.00.08 “Теория и методика профессионального образования” / К. М. Дурай-Новакова. – М., 1983. – 32 с.
76. Дьяченко М. И. Психологические проблемы готовности к деятельности / М. И. Дьяченко, Л. А. Кандыбович. – Минск: БГУ, 1976. – 176 с.
77. Дьяченко М. И. Психология высшей школы / М. И. Дьяченко, Л. А. Кандыбович, С. Л. Кандыбович. – Минск: Харвест, 2006. – 416 с.
78. Ершов А. П. Избранные труды / А. П. Ершов. – Новосибирск: Наука, 1994. – 416 с.
79. Ершов А. П. Программирование – вторая грамотность [Электронный ресурс] / А. П. Ершов // Архив академика А.П. Ершова. – Режим доступа: http://ershov.iis.nsk.su/russian/second_literacy/article.html (12.09.2011).
80. Жалдак М. І. Використання комп’ютера у навчальному процесі має бути педагогічно виваженим / М. І. Жалдак // Комп’ютер у школі та сім’ї. – 2011. – № 3. – С. 3-12.
81. Жалдак М. И. Система подготовки учителя к использованию информационной технологии в учебном процессе : дис. ... в форме науч. докл. доктора педаг. наук : 13.00.02 / М. И. Жалдак ; АПН СССР; НИИ содержания и методов обучения. – М., 1989. – 48 с.
82. Жалдак М. І. Изучение языков программирования в школе. / М. І. Шкіль, М. І. Жалдак, Н. В. Морзе, Ю. С. Рамський. // – Київ.: «Радянська школа». 1988. – 272 с.
83. Жалдак М. І. Інформатика: Навчальний посібник / М. І. Жалдак, Ю. С. Рамський / [за ред. Шкіля М. І.] – К.: Вища школа, 1991. – 319 с.
84. Жалдак М. І. Комп’ютер на уроках геометрії : посіб. для вчителів / М. І. Жалдак, О. В. Вітюк. – К.: ДІНІТ, 2003. – 168 с.
85. Жалдак М. І. Математика (тригонометрія, геометрія, елементи стохастики) з

комп'ютерною підтримкою: навч. посіб. / М. І. Жалдак, А. В. Грохольська, О. Б. Жильцов. – К.: Міжрегіон. акад. управління персоналом, 2004. – 456 с.

86. Жалдак М. І. Модель системи соціально-професійних компетентностей вчителя інформатики / М. І. Жалдак, Ю. С. Рамський, М. В. Рафальська // Науковий часопис НПУ імені М.П. Драгоманова. Серія №2: Комп'ютерно-орієнтовані системи навчання. – К.: НПУ імені М.П. Драгоманова, 2009. – № 7(14). – С. 3-10.
87. Жалдак М. І. Педагогічний потенціал комп'ютерно-орієнтованих систем навчання математики / М. І. Жалдак // Комп'ютерно-орієнтовані системи навчання : зб. наук. праць. – К.: НПУ імені М. П. Драгоманова, 2003. – Вип. 7. – С. 3-16.
88. Жалдак М. І. Чисельні методи математики : посіб. для самоосвіти вчителів / М. І. Жалдак, Ю. С. Рамський. – К. : Рад. шк., 1984. – 206 с.
89. Жоголев Е. А. Технологические основы модульного программирования / Е.А. Жоголев // Программирование. – 1980. – № 2. – с.44-49.
90. Жужжалов В. Е. Интеграционные методы изучения программирования в вузовском курсе інформатики / В. Е. Жужжалов // Вестник МГПУ. Серия: информатика и информатизация образования. – М., 2003. – № 1. – С. 53-54.
91. Жужжалов В. Е. Специфика обучения программированию при подготовке студентов – информатиков / В. Е. Жужжалов // Вестник МГЛУ. Серия Информатизация образования. –М.: 2004. – № 1(2). – С. 30-41.
92. Жук Ю. О. Концепція створення засобів навчання нового покоління для середніх закладів освіти України / Савченко О. Я., Гуржій А. М., Доній В. М., Волинський В. П., Жук Ю. О., Самсонов В. В., Шут М. І., // Проблеми освіти : наук.-метод. зб. – Київ, 1997. – Вип. 10. – С. 207-218.
93. Завадський І. О. Інформатика. Навчальна програма для 10-11 класів загальноосвітніх навчальних закладів. Рівень стандарту / І. О. Завадський, Ю. О. Дорошенко, Ж. В. Потапова // Інформатика та інформаційні технології в навчальних закладах. – 2010. – №5-6. – С. 20-33.

94. Завадський І. О. Інформатика. Навчальна програма для 10-11 класів загальноосвітніх навчальних закладів. Академічний рівень / І. О. Завадський, Ю. О. Дорошенко, Ж. В. Потапова // Інформатика та інформаційні технології в навчальних закладах. – 2010. – № 5-6. – С. 34–50.
95. Завадський І. О. Основи візуального програмування / І. О. Завадський, Р. І. Заболотний : [Навч. Посіб.]. – К.: Вид. група BHV. – 2008. – 272 с.: іл.
96. Закон України «Про вищу освіту» / Верховна Рада України, Ін-т законодавства. – К., 2002. – 96 с.
97. Зарецька І. Т. Інформатика: Навч. посібн. для 10–11 кл. середн, загально-освітн. шкіл / І. Т. Зарецька, Б. Г. Колодяжний, А. М. Гуржій, О. Ю. Соколов. – К.: Навчальна книга, 2002.– 496 с: іл. ISBN 966-7943-10-0.
98. Захарова Т. Б. Дифференциация обучения информатике в российской школе / Т. Б. Захарова // Образование граждан мира: Сб. тез. докл. Международной конференции (г.Москва, 30 сентября 4 октября 1996 г.). – М. 1996. – С. 46-47.
99. Зеленяк О. П. Практикум программирования на Turbo Pascal. Задачи, алгоритмы и решения. / О. П. Зеленяк – К.: Издательство «Диасофт», 2001. – 320 с.
100. Зубкова Т. М. Технология разработки программного обеспечения: Учебное пособие / Т. М. Зубкова. – Оренбург: ГОУ ОГУ, 2004. – 101 с.
101. Иванова Н. В. Использование принципа аналогии при обучении программированию / Н. В. Иванова, Лю Минь // Информатизация образования. 2008. Интеграция информационных и педагогических технологий: материалы междунар. науч. конф., (г. Минск, 22-25 октября 2008 г.) / редкол.: И. А. Новик (отв. ред.) [и др.]. – Минск, 2008. – С. 223-226.
102. Ігнатенко М. Я. Активізація навчально-пізнавальної діяльності учнів старших класів при вивченні математики : дис. ... доктора пед. наук: 13.00.02 / М. Я. Ігнатенко – К., 1997. – 355 с.
103. Ільченко А. А. Організація самостійної роботи майбутніх фахівців з програмування у вищих навчальних закладах І–ІІ рівнів акредитації: дис. … кандидата пед. наук : 13.00.04 «теорія і методика професійної освіти » / Алла

Анатоліївна Ільченко / Національний авіаційний університет, – К., 2011. – 310 с.

104. Інформатика. Програми для загальноосвітніх навчальних закладів / за ред. акад. М. І. Жалдака. – Запоріжжя: Прем'єр, 2003. – 304 с.
105. Кабанова-Меллер Е. Н. Учебная деятельность и развивающее образование / Е. Н. Кабанова-Меллер. – М.: Знание, 1981. – 96 с.
106. Караванова Т. П. Навчальна програма поглибленого вивчення інформатики для учнів 8-12 класів ЗНЗ (напрям: технологічний, профіль: інформаційно-технологічний) / Т. П. Караванова, В. П. Костюков // Інформатика та інформаційні технології в навчальних закладах. – 2008. – №2. – С. 5-11.
107. Караванова Т. П. Основи алгоритмізації та програмування. 750 задач з рекомендаціями та прикладами. Посібник. – К.: ТОВ. «Форум». – 2001. – 286 с.
108. Кауфман В. Ш. Языки программирования. Концепции и принципы / В. Ш. Кауфман. – М.: Радио и связь, 1993. – 432 с.
109. Кирсанов А. А. Индивидуализация учебной деятельности как педагогическая проблема / А. А. Кирсанов. – Казань: Из-во Казанского университета, 1982. – 224 с.
110. Кірей К. О. До проблеми стандартизації термінології освітніх інформаційно-елекомунікаційних технологій [Електронний ресурс] / К. О. Кірей, Л. О. Кірей // е-журнал «Педагогічна наука: історія, теорія, практика, тенденції розвитку». – 2009 – № 1. – Режим доступу: http://www.intellect-invest.org.ua/rus/pedagog_editions_e-magazine_pedagogical_science_arhiv_pn_n1_2009_st_14 (20.02.2009)
111. Клочко В. І. Застосування новітніх інформаційних технологій при вивченні вищої математики у технічному вузі : навч.-метод. посіб. / В. І. Клочко. – Вінниця : ВДТУ, 1997. – 300 с.
112. Книга вчителя інформатики: Довідково-методичне видання / Упоряд. Н. С. Прокопенко, Т. Г. Проценко – Харків: ТОРСІНГ ПЛЮС, 2005.– 256с.

113. Кнут Д. Э. Искусство программирования, том 1. Основные алгоритмы / Д. Э. Кнут. – [3-е изд.]. – [пер. с англ.]. – М.: Издательский дом «Вильямс», 2000. – 720 с.
114. Коберник О. М. Формування у студентів готовності до впровадження інноваційних педагогічних технологій / О. Коберник // Педагогіка і психологія професійної освіти. – 2002. – № 4. – С. 104-109.
115. Кобильник Т. П. Методична система навчання математичної інформатики у педагогічному університеті : дис. ... кандидата пед. наук : 13.00.02 «теорія та методика навчання (інформатика)» / Тарас Петрович Кобильник; НПУ ім. М. П. Драгоманова. – К., 2009. – 250 с.
116. Кобильник Т.П. Використання міжпредметних зв'язків при навчанні математичної інформатики у педагогічному університеті / Т.П. Кобильник // Науковий часопис НПУ імені М.П.Драгоманова. Серія № 2. Комп'ютерно-орієнтовані системи навчання: зб. наук. праць / Редрада. – К.: НПУ імені М.П. Драгоманова, 2010. – № 8 (15). – С. 143–148. (фахове видання).
117. Кобильник Т.П. Програмне забезпечення спеціального призначення у навчанні студентів інформатичних спеціальностей педагогічного університету / Т.П. Кобильник // Інформаційні технології і засоби навчання: Електронне наукове фахове видання. – 2011. – Том 23. – № 3 – Режим доступу : http://www.journal.iitta.gov.ua (фахове видання).
118. Кондрашова Л. В. Методика подготовки будущего учителя к педагогическому взаимодействию с учащимися: Учебное пособие для студуденоа пединститутов / Л. В. Кондрашова. – М., 1990. – 160 с.
119. Копаєв О. В. Модельна сутність алгоритму / О. В. Копаєв // Комп'ютерно-орієнтовані системи навчання: збірник наукових праць. – К.: НПУ імені М. П. Драгоманова, 2007. – Вип. 5 (12). – С. 189 – 193.
120. Костюченко А. А. Выбор критериев качества педагогических программных средств, для их экспертной оценки / А. А. Костюченко // Журнал научных публикаций аспирантов и докторантов. – Курск,2013. – № 9(87). – С. 216-220.

121. Костюченко А. А. Интегрированная середа разработки Lazarus, как свободно распространяемый инструмент для создания педагогических программных средств с графическим интерфейсом / А. А. Костюченко // Журнал научных публикаций аспирантов и докторантов. – Курск,2013. – № 6(84). – С. 149-151.
122. Костюченко А. О. Вимоги до педагогічних програмних засобів / А. О. Костюченко // Вісник Чернігівського державного педагогічного університету імені Т. Г. Шевченка. Серія: Педагогічні науки. – 2008. – Вип. 58. – С. 39-43.
123. Костюченко А. О. Комп'ютерний підручник як один з видів комп'ютерних засобів навчання / А. О. Костюченко // Вісник Чернігівського державного педагогічного університету імені Т. Г. Шевченка. Серія: Педагогічні науки. – 2006. – Вип. 42. – С. 102-106.
124. Костюченко А. О. Методичні основи створення педагогічних програмних засобів / А. О. Костюченко // Науковий часопис Національного педагогічного університету імені М. П. Драгоманова. Серія 2: Комп'ютерно-орієнтовані системи навчання. – К.: НПУ імені М. П. Драгоманова, 2008. – № 6(13). – С. 80-86.
125. Костюченко А. О. Методичні підходи до викладання програмування в педагогічних ВНЗ, як один з початкових кроків до створення вільнорозповсюджуваних програмних засобів навчального призначення / А. О. Костюченко // Вісник Чернігівського національного педагогічного університету імені Т. Г.Шнвченка. Серія: Педагогічні науки. – 2012. – Т. 1., Вип. 104. – С. 159-163.
126. Костюченко А. О. Методичні підходи до викладання програмування в педагогічних ВНЗ, як один з початкових кроків до створення вільнорозповсюджуваних програмних засобів навчального призначення / Костюченко А. О. // Інформаційні технології як інноваційний шлях розвитку України у XXI столітті: матеріали І Всеукраїнської науково-пратична конференція молодих науковців (м. Ужгород, Закарпатський державний університет, 06-08 грудня 2012 р.) / За ред. Ф. Г. Ващука, О. М.Ващук, І. Ф.Повхана. – Ужгород: ЗакДУ, 2013. – С. 73-76.

127. Костюченко А. О. Підходи та етапи проектування педагогічних програмних засобів / Костюченко А. О. // Стратегия качества в промышленности и образовании: материалы IX международной конференции (г. Варна, Болгария, 31 мая – 7 июня 2013 г.) Т. 2 / Упорядники: Хохлова Т. С., Хохлов В. О., Ступак Ю. О. – Днепропетровск-Варна, 2013. – С. 423-426.
128. Костюченко А. О. Про кількість нерухомих точок перестановок, число е та індивідуальний підхід у навчанні елементів стохастики майбутніх вчителів математики / А. О. Костюченко, Г. О. Михалін, С. Л. Надточій, // Науковий часопис Національного педагогічного університету імені М. П. Драгоманова. Серія 2: Комп'ютерно-орієнтовані системи навчання. – К.: НПУ імені М. П. Драгоманова, 2009. – № 7(14). – С. 118-127.
129. Костюченко А. О. Про спецкурс "Сучасні технології контролю знань, вмінь та навичок учнів на уроках інформатики" / Костюченко А. О., Пеньков А. В. // Нові інформаційні технології в навчальних закладах України: матеріали міжнародної конференції пам'яті проф. І. І.Мархеля (м. Одеса, Одеській національний морський університет, 21-26 червня 2005 р.). – Одесса: Астропринт, 2005. – С.88-89.
130. Костюченко А. О. Психолого-педагогічні складові готовності майбутніх викладачів математики та інформатики до розробки ППЗ / А. О. Костюченко // Вісник Чернігівського державного педагогічного університету імені Т. Г. Шевченка. Серія: Педагогічні науки. – 2011. – Вип. 93. – С. 74-78.
131. Костюченко А. О. Створення програмних засобів з графічним інтерфейсом / Костюченко А. О. // Інформаційні та моделюючі технології: матеріали всеукраїнської науково-практичної конференція ІМТ-2013 (м. Черкаси, 17-19 травня 2013 р.). – Черкаси, 2013. – С. 5-6.
132. Костюченко А. О. Що потрібно знати програмістові , який працює з комп'ютерною побудовою графіків функцій / А. О. Костюченко // Науковий часопис Національного педагогічного університету імені М. П. Драгоманова. Серія 2: Комп'ютерно-орієнтовані системи навчання. – К.: НПУ імені М. П. Драгоманова, 2010. – №8 (15). – С. 190-195.

133. Крамар Ю. М. Методы и информационно-программные средства синтеза и анализа стилей программирования: дис... канд. техн. наук: 05.13.06 / Юлия Михайловна Крамар / Інститут проблем моделювання в енергетиці ім. Г.Є. Пухова, Київ, 2005. – 200 с.
134. Крамаренко Т. Г. Формування особистісних якостей школяра у процесі комп'ютерно-орієнтованого навчання математики: дис. ... канд. педаг. наук : 13.00.02 / Т. Г. Крамаренко ; НПУ ім. М. П. Драгоманова. – Київ, 2008. – 270 с.
135. Кремень В. Г. Вища освіта і наука – пріоритетні сфери розвитку суспільства у XXI столітті / В. Г. Кремень // Вища шк. – 2002. – № 4-5. – С. 3-33.
136. Кривонос О. М. Особливості викладання програмування у вищому начальному закладі з врахуванням вимог сучасності [Електронний ресурс] / О. М. Кривонос // Електронна бібліотека Житомирського державного університету. – Режим доступу: http://eprints.zu.edu.ua/5263/1/vip_57_27.pdf (18.05. 2011).
137. Криницкий Н. А. Алгоритмы вокруг нас / Н. А. Криницкий – М.: Наука, 1977. – 224 с.
138. Кузнєцов А. А. Принципи диференціації змісту навчання інформатики / А. А. Кузнєцов, Т. Б. Захарова // Інформатика та освіта. – 1997. – № 4. – С. 13-18.
139. Кузьмина Н. В. Методы исследования педагогической деятельности / Н. В. Кузьмина. – Л.: Изд-во ЛГУ, 1970. – 211 с.
140. Кузьмінський А. І. Технологія і техніка шкільного уроку: Навч. посіб. Рекомендовано МОН / А. І. Кузьмінський, С. В. Омельяненко. – К., 2010. – 335 с.
141. Лаврентьєва Г.П. Психолого-ергономічні вимоги до застосування електронних засобів навчання: [Електронний ресурс] / Г.П. Лаврентьєва // Національна бібліотека України імені В. І. Вернадського. – Режим доступу: http://archive.nbuv.gov.ua/e-journals/ITZN/em12/content/09lgpeom.htm. – Назва з екрану.

142. Лапінський В. В. Принцип наочності і створення електронних засобів навчального призначення [Електронний ресурс] / В. В. Лапінський // Національна бібліотека України імені В. І. Вернадського. – Режим доступу: http://www.nbuv.gov.ua/e-journals/NarOsv/2009-3/9lvvznp.htm
143. Лекции лауреатов премии Тьюринга за первые двадцать лет, 1966-1985 / [пер. с англ.]; под ред. Эшенхёрста Р. – М.: Мир, 1993. – 560 с., ил.
144. Лернер И. Я. Дидактические основы методов обучения / И. Я. Лернер. –М.: Педагогика, 1981. – 186 с.
145. Лесневский А. С. Об основных понятиях школьного курса информатики. / А. С. Лесневский. / – Информатика и образование. – 1994. – №2. – С. 41-44.
146. Линенко А. Ф. Теория и практика формирования готовности студентов педагогических вузов к профессиональной деятельности: дис. … докора пед. наук: 13.00.01, 13.00.04 / А. Ф. Линенко. – К., 1996. – 378 с.
147. Лисенко Т. І. Інформатика 11 : підруч. [для заг. навч. закл. ; академ.] / Й. Я. Ривкінд, В. В. Шакотько, Т. І. Лисенко, Л. А. Чернікова – К.: Генеза – 2011 – 304 с. : іл.
148. Ліннік О. П. Об’єктно-орієнтоване моделювання у підготовці майбутніх учителів фізики / О. П. Ліннік, Н. В. Моісеєнко, В. М. Євтєєв та ін. // Збірник наукових праць Кам’янець-Подільського державного університету. Серія педагогічна: Проблеми дидактики фізики та шкільного підручника фізики в світлі сучасної освітньої парадигми. – Кам’янець-Подільський, 2006. – Вип. 12.– С. 127-130.
149. Лобко-Лобановская Н. А. Дифференцированное обучение как способ формирования познавательной активности школьников: дис. ... канд. пед. наук: 13.00.01 / Н. А. Лобко-Лобановская; Харьковский государственный педагогический инстиут им. Г. С. Сковороды. – Харьков, 1991. – 242 с.
150. Логические основы ЭВМ. Представление команд в ЭВМ [Электронный ресурс] // Веб-сайт «html-kod.ru». – Режим доступа: http://html-kod.ru/docs/index-1827.html (19.03.2013). – Заглавие с экрана.
151. Любченко К. М. Елементи математичної логіки з комп’ютерною підтримкою

: посіб. для вчителів / К. М. Любченко, Ю. В. Триус. – Черкаси : Черкас. нац. ун-т ім. Б. Хмельницького, 2004. – 87 с.

152. Ляшенко О.І. Диференціація як основоположний принцип шкільного навчання / О. І. Ляшенко // Педагогіка і психологія. – 2009. – № 1. – С. 40-45.
153. Майерс Г. Надежность программного обеспечения / Г. Майерс. – М.: Мир, 1980. – 359 с.
154. Максименко С. Д. Диференційоване навчання: До проблеми психологічного супроводу / С. Д. Максименко // Педагогіка і психологія. – 2009. – № 1. – С. 46-53.
155. Мануйлов В. Г. Разработка программного обеспечения на Паскале: учеб. пособие / В. Г. Мануйлов; под ред. А. И. Китова – М.: ПРИОР, 1996. – 240 с., ил.
156. Машбиц Е. И. Психолого-педагогические проблемы компьютеризации обучения / Е. И. Машбиц. – М.: Педагогика, 1988. – 192 с.
157. Меджитова Л.М. Підхід до будування вступного курсу програмування для студентів комп'ютерних спеціальностей / Л. М. Меджитова // Науковий часопис Національного педагогічного університету імені М. П. Драгоманова. Серія 2, Комп'ютерно-орієнтовані системи навчання. – К.: НПУ ім. М. П. Драгоманова, 2011. – Вип. 11 (18). – С. 51-56.
158. Мичтом Дж. Структурный подход к программированию / Дж. Хьюз, Дж. Мичтом; [пер. с англ. Э. М. Киуру и А. Л. Александрова]. – М.: Мир, 1980. – С. 29-71.
159. Монахов В. М. Технологические основы проектирования и конструирования учебного процесса / В. М. Монахов. – Волгоград : Перемена, 1995. – 152 с.
160. Морзе Н. В. Методика навчання інформатики. Ч. 4. Методика навчання основам алгоритмізації і програмування / Н. В. Морзе. – К.: Навч. кн., 2003. – 250 с.
161. Морзе Н. В. Інформатика : підруч. для 11 кл. загальноосвіт. навч. закл.: рівень стандарту / Н. В. Морзе, О. В. Барна, В. П. Вембер, О. Г. Кузьмінська, – К.: Школяр 2011. – 304с.: іл.

162. Морзе Н. В. Система методичної підготовки майбутніх вчителів інформатики в педагогічних університетах: дис. ... доктора пед. наук: 13.00.02 / Наталія Вікторівна Морзе. – К.: НПУ імені М. П. Драгоманова., 2003. – 600 с.: іл.
163. Мухортов В. В. Объектно-ориентированное программирование, анализ и дизайн: Методическое пособие / В. В. Мухортов, В. Ю. Рылов – Новосибирск : Новософт, 2002. – 108 с.
164. Навчальна програма з математики для загальноосвітніх навчальних закладів, 10-11 класи (академічний рівень) // Математика в школі. – 2011. – № 6. – С. 11-20.
165. Навчальна програма з математики для загальноосвітніх навчальних закладів, 10-11 класи (профільний рівень) // Математика в школі. – 2011. – № 7-8. – С. 3-16.
166. Новик М. М. Концепция создания лаборатории активных методов обучения в Центре прогрессивных технологий обучения СПбГИЭА / М. М. Новик // Активные методы обучения в системе многоуровнего образования: сборник научных трудов. – СПб.: СПбГИЭА, 1995. – С. 45-47.
167. О системе программирования PascalABC.NET. [Електронний ресурс] // Веб-сайт «PascalABC.NET». – Режим доступа: http://pascalabc.mmcs.rsu.ru. – Заглавие с экрана
168. О. Г. Абрамова Психолого-педагогические проблемы дифференцированного обучения / О. Г. Абрамова, И. С. Якиманская // Советская педагогика. – 1991. – № 4. – С. 44-52.
169. Онищенко С. М. Вивчення основ програмування на базі систем програмування Turbo Pascal та Delphi в середній загальноосвітній школі / С. М. Онищенко // Комп’ютерно-орієнтовані системи навчання: зб. наук. пр. НПУ ім. М. П. Драгоманова. – К.: НПУ, 2000. – Вип.2. – С. 163 – 167.
170. Онищенко С. М. Використання Delphi (Kylix) при вивченні основ програмування / С. М. Онищенко, В. М. Франчук // Вісник: Зб. наук. статей НПУ ім. М. П. Драгоманова. – К.: НПУ ім. М. П. Драгоманова, 2004. Вип.6

171. Осмоловская И. Практика дифференцированного обучения: попытка систематизации / И. Осмоловская // Школа. – 1996. – № 6. – С. 45-50.
172. Пайл Я. АДА – язык встроенных систем / Я. Пайл; [пер. с англ.]. – М.: Финансы и статистика, 1984. – 238с.
173. Парадигма програмування [Електронний ресурс] // Вікіпедія – вільна енциклопедія. – Режим доступу: http://uk.wikipedia.org/wiki/Парадигма_програмування (02.04-2013). – Назва з екрану.
174. Педагогика: Учебное пособие для студентов пед. вузов / [Бабанский Ю. К., Мластенин В. А., Сорокин Н. А. и др.] ; под ред. Ю. К. Бабанского. – [2-е изд., доп. и перераб.]. – М. : Просвещение, 1988. – 479 с.
175. Пеньков А. В. Использование новой информационной технологии при преподавании математики в старших классах средней школы : дис … канд. пед. наук : 13.00.02 / А. В. Пеньков. – К., 1992. – 171 с.
176. Петров А. Н. Совершенствование методики обучения объектно-ориентированному программированию на основе объектно-ориентированного проектирования: на примере дисциплины "Программирование" для будущих учителей информатики : автореф. дис. на соискание уч. степени канд. пед. наук : спец. 13.00.02 – теория и методика обучения и воспитания (информатика, уровень высшего профессионального образования)/ А. Н. Петров – Москва – 2009. – 20 с.
177. Підготовка учнів до професійного навчання і праці (психолого-педагогічні основи): Навч. посібник / Під ред. Г. О. Балла, П. С. Перепелиці, В. В. Рибалки. – К.: Наукова думка, 2000. – 188 с.
178. Пойя Д. Как решать задачу / Д Пойя; [пер. с англ.]. – М.: Либроком, 2010. – 208 с.
179. Полищук А. П. Методы вычислений в классах языка С++: учеб. пособие / А. П. Полищук, С. А. Семериков. – Кривой Рог : Издательский отдел КГПИ, 1999. – 350 с.
180. ППЗ Gran2D [Электронный ресурс]: середовище динамічної геометрії

призначене для створення динамічних геометричних побудов на площині та графічного аналізу створених побудов: [програмний засіб]. – 700 kB / M.I. Жалдак, Ю.В. Горошко, А.О.Костюченко. – Систем. вимоги: Pentium 100 MHz; 32 Mb RAM; 8 Mb Video; Windows 95/98/ME/XP/2000. – Режим доступу: http://fizmat.chnpu.edu.ua/gran2d (11.11.2012).– Назва з екрану.

181. ППЗ Numet [Электронный ресурс]: підтримка викладання чисельних методів: [програмний засіб]. – 390 kB / А.О. Костюченко. – Систем. вимоги: Pentium 100 MHz; 32 Mb RAM; 8 Mb Video; Windows 95/98/ME/XP/2000. – Режим доступу: http://fizmat.chnpu.edu.ua/numet (09.01.2007).– Назва з екрану.
182. Про затвердження Державної програми "Інформаційні та комунікаційні технології в освіті і науці" на 2006-2010 роки : постанова Кабінету Міністрів України // Офіційний вісник України. – 2005. – № 49. – С. 40.
183. Про затвердження тимчасових вимог до педагогічних програмних засобів (Наказ МОН України № 360 від 15.05.2006) [Електронний ресурс]. – Режим доступу: http://zakon.nau.ua/doc/?uid=1038.1132.0). – Назва з екрану.
184. Про Національну доктрину розвитку освіти : Указ Президента України // Законодавчі акти України з питань освіти. – К.: Парламент. вид-во, 2004.
185. Про невідкладні заходи щодо забезпечення функціонування та розвитку освіти в Україні (Указ Президента України № 1013/2005 від 04.06.05р.) // Збірник нормативно-правових документів з вищої освіти. – К., 2007. – 87 с.
186. Про основні засади розвитку інформаційного суспільства в Україні на 2007-2015 роки (Закон України № 537-V від 9 січня 2007 року) [Електронний ресурс] // Відомості Верховної Ради України (ВВР). – 2007. – №12. – С.102. – Режим доступу: http://zakon4.rada.gov.ua/laws/show/537-16.
187. Програмування в обмеженнях [Електронний ресурс] // Вікіпедія – вільна енциклопедія. – Режим доступу: http://uk.wikipedia.org/wiki/Програмування_в_обмеженнях (27.03.2013). – Назва з екрану.
188. Раков С. А. Дослідницький підхід у математичній освіті, пакети динамічної геометрії та динамічні опорні конспекти / С. А. Раков // Комп’ютер у школі і

сім'ї. – 2005. – № 5. – С. 17-21.

189. Раков С. А. Компьютерные эксперименты в геометрии / С. А. Раков, В. П. Горох. – Х. : РЦНИТ. – 1996. – 176 с.
190. Раков С. А. Математична освіта : компетентнісний підхід з використанням ІКТ : монографія / С. А. Раков. – Х. : Факт, 2005. – 360 с.
191. Раков С. А. Відкриття геометрії через комп'ютерні експерименти в пакеті DG: Посібник для вчителів математики / [авт. кол.: С. А. Раков, В. П. Горох, К. О. Осенков, та ін.]. – Харків : Вікторія, 2002. – 136 с.
192. Рамський Ю. С. Формування інформаційної культури вчителя математики при вивченні методів обчислень у педагогічному вузі / Ю. С. Рамський // Комп'ютерно-орієнтовані системи навчання : збірник наукових праць. – К., 2000. – Вип. 2. – С. 25-47.
193. Рамський Ю. С. Вивчення Web-програмування в школі: Навчальний посібник. / Ю. С. Рамський, І. С. Іваськів, О. Ю. Ніколаєнко – Тернопіль: Навчальна книга – Богдан , 2004. – 200 с.
194. Рамський Ю. С. Методика навчання основ об'єктно-орієнтованого програмування / Рамський Ю. С., Лукаш І. М. //Комп'ютер у школі та сім'ї. – 2002. – №1. – С. 3-7; №2. – С. 3-8 ; №3. – С. 7-13; №4. – С. 17-22; №5. – С. 10-17; №6. – С.16.-21.
195. Рамський Ю. С. Методика навчання основ об'єктно-орієнтованого програмування. Об'єктно-орієнтований аналіз / Рамський Ю. С., Лукаш І. М. // Комп'ютер у школі та сім'ї. – 2003. – №1. – С.3-9.
196. Рамський Ю. С. Основи програмування (мовою Паскаль) : короткий курс лекцій : лаборатор. практикум : [навч. посіб. для студ.] / Ю. С .Рамський, Г. Ю. Цибко. – К.: НПУ ім. М. П.Драгоманова, 2004. – 141 с.
197. Резанович А. Е. Развитие готовности студентов вузов к организаторской деятельности: автореф. дисс. на соискание уч. степени канд. пед. наук : спец. 13.00.08 «Теория и методика проф. образ.» / А. Е. Резанович. – Магнитогорск, 2002. – 22 с.

198. Рубинштейн С. Л. Основы педагогической психологии / С. Л. Рубинштейн. – СПб, 1999. – 720 с.
199. Руденко В. Д., Базовий курс інформатики / В. Д. Руденко, О. М. Макарчук, М. О. Патланжоглу; [за заг. ред. В. Ю. Бикова] : [навч. посіб.] – К.: Вид. група BHV. – Кн. 2: Інформаційні технології, 2006. – 368 с. іл.
200. Себеста Р. Основные концепции языков программирования / Р. Себеста / [5-е издание] : [пер. с англ.] – М.: Издательский дом «Вильямс», 2001. – 672 с.
201. Сейдаметова З. С. Методическая система уровневой подготовки будущих инженеров-программистов по специальности «Информатика» : дис. ... доктора пед. наук : 13.00.02 / Зарема Сейдалиевна Сейдаметова; НПУ им. М. П. Драгоманова. – К., 2007. – 559 с.
202. Семеріков С. О. Активізація пізнавальної діяльності студентів при вивченні чисельних методів у об’єктно-орієнтованій технології програмування. Автореф. дисс. на соискание ученой степени канд. пед. наук: 13.00.02. – теория и методика обучения информатике. / С. О. Семеріков – Национальный педагогический университет имени М. П. Драгоманова, Киев, 2001. – 20 с.
203. Семеріков С. О. Вільне програмне забезпечення як фактор стабілізації вузівських курсів інформатики / С. О. Семеріков, І. О. Теплицький // Інформаційні технології в освіті : матеріали Всеукраїнської наук.-практ. конф., 24–26 травня 2006 р. – Мелітополь, 2006. – С. 55–56.
204. Семеріков С. О. Еволюція та сучасний стан курсу чисельних методів у вищій школі / С. О. Семеріков // Збірник наукових праць Кам’янець-Подільського державного педагогічного університету : Серія педагогічна : Дидактики дисциплін фізико-математичної та технологічної освітніх галузей. – Кам’янець-Подільський, 2002. – Вип. 8. – С. 189-193.
205. Семеріков С. О. Теоретико-методичні основи фундаменталізації навчання інформатичних дисциплін у вищих навчальних закладах: дис. ... доктора педаг. наук: 13.00.02 / С. О. Семеріков ; – К.: НПУ ім. Драгоманова, 2009. – 522 с.

206. Семеріков С. О. , Роль, місце та зміст комп’ютерного моделювання в системі шкільної освіти // Семеріков С. О., Теплицький І. О. // Науковий часопис НПУ ім. М. П. Драгоманова. Серія №2. Комп’ютерно-орієнтовані системи навчання. Зб.наукових праць / Ред.рада. – К.: НПУ ім. М. П. Драгоманова, 2010. № 9(16). – С. 3 – 11.
207. Скафа Е.И. Эвристическое обучение математике: теория, методика, технология: монография / Е.И.Скафа. – Донецк: Изд-во ДонНУ, 2004. – 440 с.
208. Сластенин В. А. Формирование личности учителя советской школы в процессе профессиональной підготовки / В. А. Сластенин. – М.: Просвещение, 1987. – 159 с.
209. Слєпкань З. І. Ще раз про диференціацію навчання і роль в ній освітнього стандарту / З. І. Слєпкань // Математика в школі. – 2002. – № 2. – С. 29-30.
210. Смирнова-Трибульска Є. М. Інформаційно-комунікаційні технології в професійній діяльності вчителя: посібник для вчителів / Є. М. Смирнова-Трибульска ; науковий редактор д.пед.н., академік АПН України, проф. М. І. Жалдак. – Херсон : Айлант, 2007. – 560 с.
211. Спиваковский А. В. Web-среда для изучения основ алгоритмизации и программирования / А. В. Спиваковский, Н. В. Колесникова, Н. И. Ткачук, И. М. Ткачук // Управляющие системы и машины. – К., 2008. – С. 70-75.
212. Співаковський О.В Шляхи удосконалення курсу “Основи алгоритмізації та програмування” у педагогічному вузі. / О.В. Співаковський, М.С. Львов // Комп’ютер у школі та сім’ї. – 2001. – №4. – С. 22-24.
213. Співаковський О.В. Концепція викладання дисциплін інформатики в школі і педагогічному вузі / Співаковський О.В. // Комп’ютер у школі та сім’ї. – 2003. – № 3. – С. 21-25
214. Співаковський О. В. Основи програмування. Навчальний посібник. / А. М. Гуржій, М. С. Львов , О. В. Співаковський – К.: Наукова думка, 2004. – 355 с.

215. Співаковський О. В. Теорія й практика використання інформаційних технологій у процесі підготовки студентів математичних спеціальностей: Монографія / О. В. Співаковський. – Херсон: Айлант. – 2003. – 229 с.
216. Спірін О. М. Початки штучного інтелекту : навч. посіб. для студ. фіз.-мат. спец. вищ. пед. навч. закл. / О. М. Спірін – Житомир : Вид-во ЖДУ, 2004. – 172 с.
217. Спірін О. М. Система інформаційно-технологічних компетентностей учителя інформатики / О. М. Спірін // Інформаційно-комунікаційні технології навчання: матеріали міжнар. наук.-практ. конференції. – Умань: ПП Жовтий, 2008. – С. 160-162.
218. Спірін О. М. Диференційований підхід у вивченні основ штучного інтелекту в курсі інформатики фізико-математичного факультету вищого педагогічного закладу: дис. … кандата пед. наук : 13.00.02 / Олег Михайлович Спірін; НПУ ім. Драгоманова. – К., 2001. – 223 с.
219. Степанов А. Г. Объектно-ориентированная модель информатики как предмета обучения [Электронный ресурс] / А. Г. Степанов // Материалы конференции ИТО-2006. – М., 2006. – Режим доступу : http://ito.edu.ru/2006/Moscow/I/1/I-1-6306.html – 5.09.2012 р.
220. Степанов А. Г. Объектно-ориентированный подход к отбору содержания обучения информатике / А. Г. Степанов. – СПб. : Политехника, 2005. – 229 с.
221. Структурное программирование. Из истории программирования. [Электронный ресурс] // Информационные технологии для начинающих и не только. – Режим доступа: http://manuilov.narod.ru/structura/2_1.htm – Заглавие с экрана.
222. Татур Ю. Г. Высшее образование: методология и опыт проектирования: учеб.-методич. пособие / Ю. Г. Татур. – М.: Логос: Универ. кн., 2006. – 252 с.
223. Теплицький І. О. «Віртуальний фізичний лабораторний практикум» як актуальна проблема сучасної дидактики / І. О. Теплицький, С. О. Семеріков // Теорія та методика навчання математики, фізики, інформатики : зб. наук. праць : в 3-х т. – Кривий Ріг , 2004. – Т. 2, вип. 4 : Теорія та методика

навчання фізики. – С. 414-421.

224. Теплицький І. О. З досвіду використання Вільного програмного забезпечення у підготовці майбутнього вчителя / І. О. Теплицький, С. О. Семеріков // Рід. шк. – 2003. – № 5. – С. 40–41.
225. Теслер Г. С. Новая кибернетика / Г. С. Теслер. – К.: Логос, 2004. – 404 с.
226. Тищенко С. І. Інтегрування змісту математичних і спеціальних дисциплін у професійній підготовці молодших спеціалістів з програмування: дис. … кандидата пед. наук: 13.00.04 «теорія і методика професійної освіти » / Світлана Іванівна Тищенко / Інститут педагогічної освіти і освіти дорослих АПН України, К., 2009. – 280 с.
227. Требования к ППС [Электронный ресурс] // Веб-сайт «wiki.irkutsk.ru». – Режим доступа: http://wiki.irkutsk.ru/index.php/Требования_к_ППС (19.06.2007). – Заглавие с экрана.
228. Триус Ю. В. Комп’ютерно-орієнтовані методичні системи навчання математичних дисциплін у вищих навчальних закладах : дис. ... доктора пед. наук : 13.00.02 / Ю. В. Триус ; Черкаський нац. ун-т ім. Б. Хмельницького. – Черкаси, 2005. – 649 с.
229. Триус Ю. В. Особливості створення методичної системи навчання основ програмування для підготовки майбутніх інженерів-програмістів / Ю. В. Триус, О. О. Богатирьов, Л. В. Гришко // Вісник Черкаського університету. Серія: Педагогічні науки. – Черкаси, 2002. – Вип. 35. – С. 133-141.
230. Троелсен Эндрю. Язык программирования C# 2005 (Си Шарп) и платформа .NET 2.0 = Troelsen, Andrew. Pro C# 2005 and the .NET 2.0 Platform. / Andrew Troelsen = Эндрю Троелсен. – 3-е изд. – М.: «Вильямс», 2007. – 1168 с. – ISBN 5-8459-1124-9, 978-5-8459-1124-7.
231. Турский В. Методология программирования / В. Турский. – М.: Мир, 1981. – 272 с.
232. Український педагогічний словник / [авт. –уклад. Гончаренко С. У.]. – К.: Либідь, 1997. – 376 с.

233. Унт І. Е. Диференціація навчання на уроках інформатики /І. Е. Унт // Інформатика та освіта. – 1999. – № 1. – С.26-33.
234. Фирсов В. В. О существе уровневой дифференциации обучения [Електронний ресурс] / В. В. Фирсов // е-журнал «Педагогічна наука: історія, теорія, практика, тенденції розвитку». – 2008 – № 1. – Режим доступу: http://intellect-invest.org.ua/pedagog_editions_e-magazine_pedagogical_science_arhiv_pn_n1_2008_firsov_o_sushestve_urovnevoj_differentiatzii/
235. Фокин Р. Р. Метамодель обучения информатике в высшей школе : автореф. дис. на соискание ученой степени доктора пед. наук : 13.00.02 «Теория и методика обучения информатике» / Р. Р. Фокин. – СПб., 2000. – 32 с.
236. Фуксман А. Л. Технологические аспекты создания программных систем / А. Л. Фуксман. – М.: Статистика, 1979. – 184 С.
237. Функціональне програмування [Електронний ресурс] // Вікіпедія – вільна енциклопедія. – Режим доступу: http://uk.wikipedia.org/wiki/Функціональне_програмування (5.09.2012). – Назва з екрана.
238. Харламов И. Ф. Педагогика / И. Ф. Харламов – М.: Гардарики, 1999. – 520 с.
239. Чуканов В. О. Оценка качества программных средств учебного назначения / В. О. Чуканов, В. В. Гуров // Качество. Инновации. Образование. – 2007. – № 5. – С. 27-32.
240. Шахмаев Н. М. Учителю о дифференцированном обучении / Н. М. Шахмаев. – М.:НИИ ОП АПН СССР, 1989. – 250 с.
241. Шевчук П. Г. Значення стилю програмування в процесі навчання учнів та студентів / О. М. Кривонос, П. Г. Шевчук // Комп’ютерно-інтегровані технології: освіта, наука, виробництво : Науковий журнал; Відп. ред. В. Д. Рудь. – Луцьк, 2011. – № 5 – С. 148 – 150
242. Шевчук П. Г. Навчання об’єктно-орієнтованому програмуванню в загальноосвітніх навчальних закладах засобами мови C# / П. Г. Шевчук // Теорія та методика навчання математики, фізики, інформатики: збірник

наукових праць. Випуск IX. – Кривий ріг: Видавничий відділ НМетАУ, 2011 – С.595 – 601.

243. Шевчук П. Г. Основні підходи добору мови та середовища програмування як засобів навчання // П. Г. Шевчук / Інформаційні технології і засоби навчання: електронне наукове фахове видання [Електронний ресурс] / Ін-т інформ. технологій і засобів навчання АПН України, Ун-т менеджменту освіти АПН України; гол. ред.: В. Ю. Биков. – 2010. – № 3(17). – Режим доступу: http://journal.iitta.gov.ua/index.php/itlt/article/view/251
244. Шнейдерман Б. Психология программирования: Человеческие факторы в вычислительных и информационных системах / Б. Шнейдерман; [пер. с англ.]. – М.: Радио и связь, 1984. – 304 с.
245. Bobrow D. G. If Prolog is the answer, what is the question / D. G. Bobrow // Fifth Generation of Computer Systems (Tokyo, Japan, November 1984.) / Institute for New Generation Computer Technology (ICOT). – North-Holland, 1984. – P. 138-145.
246. Budd T. A. Multy-Paradigm Programming in LEDA / T. A. Budd. – Addison-Wesley; Reading; Massachusets, 1995. – 320 p.
247. Corrado B. Flow diagrams, turing machines and languages with only two formation rules / Bohm Corrado, Giuseppe Jacopini // Communications of the ACM. – New York, 1966. – № 9 (5) P. 366-371.
248. Friedman L. W. Comparative programming languages: generalizing the programming function / L. W. Friedman. – Prentice Hall, 1991. – 188 p. [L14.05
249. Holt R. C. Structure of Computer Programs: A Survey / R. C. Holt // Proceedings of the IEEE. – 1975. – № 63(6). – p. 879-893.
250. Jackson D. A New Approach to Teaching Programming [Электронный ресурс] / D. Jackson, R. Miller. – Режим доступа: http://people.csail.mit.edu/dnj/articles/teaching-6005.pdf. – Назва з екрану.
251. Liskov B. Data abstraction and hierarchy [Electronic Resource] / Barbara Liskov // Scientific Literature Digital Library. – 2008. – 18. p. – Mode of access:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.819&rep=rep1&type=pdf. – Title from the screen.

252. Rudder A. Teaching programming using visualization [Электронный ресурс] / A. Rudder, M. Bernard, S. Mohammed. – Режим доступа: http://sta.uwi.edu/staff/mbernard/research_files/WBE2007.pdf. – Назва з екрану
253. Shaw, M. We Can Teach Software Better / Mary Shaw // Computing Research News. – September 1992. – 4(4) – P. 2–4, 12.
254. Shriver B. Software paradigms / Bruce Shriver // IEEE Software. – 1986. – № 3(1):2. – p. 37-48.
255. Smalltalk. Объектно-ориентированное программирование. [Електронний ресурс] // Веб-сайт «Smalltalk по-русски». – Режим доступу: http://www.smalltalk.ru/articles/smalltalk.html. – Заглавие с экрана
256. Software engineering. Product quality. Part 2: External metrics: ISO/IEC TR 9126-2:2003. – [Effective as of 2004-02-21]. – American National Standards Institute, 2007. – 96 p.
257. Software engineering. Product quality. Part 3: Internal metrics: ISO/IEC TR 9126-3:2003. – [Effective as of 2004-02-21]. – American National Standards Institute, 2007. – 72p.
258. Software engineering. Product quality. Part 4: Quality in use metrics: ISO/IEC TR 9126-4:2004. – [Effective as of 2004-04-29]. – American National Standards Institute, 2007. – 66p.
259. Software engineering. Software product quality. Part 1: Quality model: ISO/IEC 9126-1:2001. – [Effective as of 2001-06-15]. – American National Standards Institute, 2007. – 32 p.
260. SOLID (об’єктно-орієнтоване програмування) [Електронний ресурс] // Вікіпедія – вільна енциклопедія. – Режим доступу: http://uk.wikipedia.org/wiki/SOLID_(об’єктно-орієнтоване_програмування) (14.03.2013). – Назва з екрану.
261. Sommerville I. Software Engineering / Ian Sommerville. – Addison-Wesley Publishing Company, 1992. – 280 p.

262. Spinellis D. Programming paradigms as object classes: a structuring mechanism for multiparadigm programming: PhD thesis / Diomidis Spinellis; University of London SW7 2BZ. – United Kingdom, 1994.
263. Wegner P. Concepts and paradigms of object-oriented programming. // ACM SIGPLAN OOPS Messenger. – 1990. – V.1, № 1.
264. Whitsed N. Learning and Teaching / Nicky Whitsed // Health Information & Libraries Journal. – 2004. – Vol. 21. – p. 273-275.

ДОДАТКИ

Додаток А.

Вихідний код класів “точка”, “коло”, “вектор” пропедевтичного курсу з ООП

```
program OOP;
{$mode objfpc}{$H+}
uses sysutils, crt;
Type
  { TDot }
 TDot=class(TObject)
  private
   Fx, Fy: Real;
   FCoord: String;
   procedure SetX(const Value: Real);
   procedure SetY(const Value: Real);
   procedure ChangInfo; virtual;
  public
   constructor create; overload;
   constructor create(Ax, Ay: Real); overload;
   procedure ShowInfo; virtual;
   procedure MoveTo(NewX, NewY: Real);
   procedure Recalc; virtual; abstract;
   property X: Real read Fx write SetX;
   property Y: Real read Fx write SetY;
   property Coord: String read FCoord;
 End;

  { TCircle }
 TCircle=class(TDot)
  private
   Fr: Real;
   FRivn: String;
   FArea: Real;
   procedure SetR(const Value: Real);
```

```
   procedure ChangInfo; override;
  public
   constructor create; overload;
   constructor create(Ax, Ay, Ar: Integer); overload;
   procedure ShowInfo; override;
   procedure Recalc; override;
   property r: Real read Fr write SetR;
   property Rivn: String read FRivn;
   property Area: Real read FArea;
 End;

  { TVector }
 TVector=class(TDot)
  private
   FLen: Real;
  public
   procedure ShowInfo; override;
   procedure Recalc; override;
   property Len: Real read FLen;
 End;

{ TDot }
constructor TDot.create;
begin
  inherited create;
  Write('Введіть координату x: ');
  Readln(Fx);
  Write('Введіть координату y: ');
  Readln(Fy);
  ChangInfo;
end;
constructor TDot.create(Ax, Ay: Real);
begin
  Fx:=Ax;
  Fy:=Ay;
  ChangInfo;
```

```
end;
procedure TDot.SetX(const Value: Real);
begin
  If Fx=Value Then Exit;
  Fx:=Value;
  ChangInfo;
end;
procedure TDot.SetY(const Value: Real);
begin
  If Fy=Value Then Exit;
  Fy:=Value;
  ChangInfo;
end;
procedure TDot.ChangInfo;
begin
  FCoord:='('+FloatToStr(Fx)+'; '+FloatToStr(Fy)+')';
end;
procedure TDot.ShowInfo;
begin
  WriteLn('Координати точки '+FCoord);
end;
procedure TDot.MoveTo(NewX, NewY: Real);
begin
  Fx:=NewX;
  Fy:=NewY;
  ChangInfo;
end;

{ TCircle }
constructor TCircle.create;
begin
  WriteLn('Введіть координати центра кола');
  inherited create;
  Write('Введіть радіус кола: ');
  Readln(Fr);
end;
```

```
constructor TCircle.create(Ax, Ay, Ar: Integer);
begin
  Fr:=Ar;
  inherited create(Ax,Ay);
end;
procedure TCircle.SetR(const Value: Real);
begin
  If Fr=Value Then Exit;
  Fr:=Value;
  ChangInfo;
end;
procedure TCircle.ChangInfo;
Var s:String;
begin
  If Fx=0 Then s:='x^2';
  If Fx<0 Then s:='(x'+FloatToStr(Fx)+')';
  If Fx>0 Then s:='(x+'+FloatToStr(Fx)+')';
  s:=s+'^2+';
  If Fy=0 Then s:=s+'y^2';
  If Fy<0 Then s:=s+'(y'+FloatToStr(Fy)+')';
  If Fy>0 Then s:=s+'(y+'+FloatToStr(Fy)+')';
  s:=s+'^2';
  s:=s+'='+FloatToStr(Fr)+'^2';
  FRivn:=s;
end;
procedure TCircle.ShowInfo;
begin
  WriteLn('Рівняння кол: '+FRivn);
  WriteLn('Площа кол: '+FloatToStr(FArea));
end;
procedure TCircle.Recalc;
begin
  FArea:=Pi*Sqr(r);
end;

{ TVector }
```

```
procedure TVector.ShowInfo;
begin
  WriteLn('Довжина вектора: '+FloatToStr(FLen));
end;
procedure TVector.Recalc;
begin
  FLen:=sqrt(sqr(x)+sqr(y));
end;

Var Dot:TDot;
    NewX, NewY: Real;
    n:Integer;
begin
  Writeln('Для створення кола введіть 1');
  Writeln('Для створення вектору введіть 2');
  Writeln('Для створення точки введіть інше число');
  Readln(n);
  If n=1
   Then Dot:=TCircle.Create
   Else If n=2 Then Dot:=TVector.Create
         Else Dot:=TDot.Create;
  Dot.ShowInfo;
  Write('Введіть нову координату x: ');
  Readln(NewX);
  Write('Введіть нову координату y: ');
  Readln(NewY);
  Dot.MoveTo(NewX, NewY);
  Dot.ShowInfo;
  Dot.Free;
  Readln;
end.
```

Додаток Б. Додаткові завдання щодо розроблення ППЗ

Розробити ППЗ, яке забезпечує користувачеві:

1. Побудову правильних многокутників, з заданої кількістю сторін (кутів). Передбачити побудову многокутника за: а) двома суміжними вершинами (проти годинникової стрілки або за годинниковою стрілкої); б) центром многокутника й однією з вершин многокутника.
2. Побудови графіків функцій у декартовій та полярній с системі координат, які задані у різних формах: а) явній формі; б) параметричній формі. Оснастити програму додатковими можливостями дослідження графіків функцій, які залежать від параметра, а також інтерактивними засобами дослідження графіків.
3. Задання статистичної вибірки відповідно до дискретної моделі даних в вигляді: частот, відносних частот чи варіант. Передбачити можливість побудови частотної таблиці (має містити: значення та частота варіант, накопичена частота, відносна частота варіант, накопичена відносна частота) та графіка щільності розподілу, що залежить від математичного сподівання та середньо квадратичного відхилення для даної вибірки.
4. Розв’язання транспортної задачі з можливістю побудови опорного плану методами: північно-західного кута, мінімальних витрат, подвійного мінімуму, Фогеля.
5. Розв’язання задачі лінійного програмування симплекс методом та можливість побудови та розв’язання двоїстої задачі до вхідної.

Додаток В.

Загальний вигляд вікон задання характеристик для створення геометричних об’єктів ППЗ Gran2d

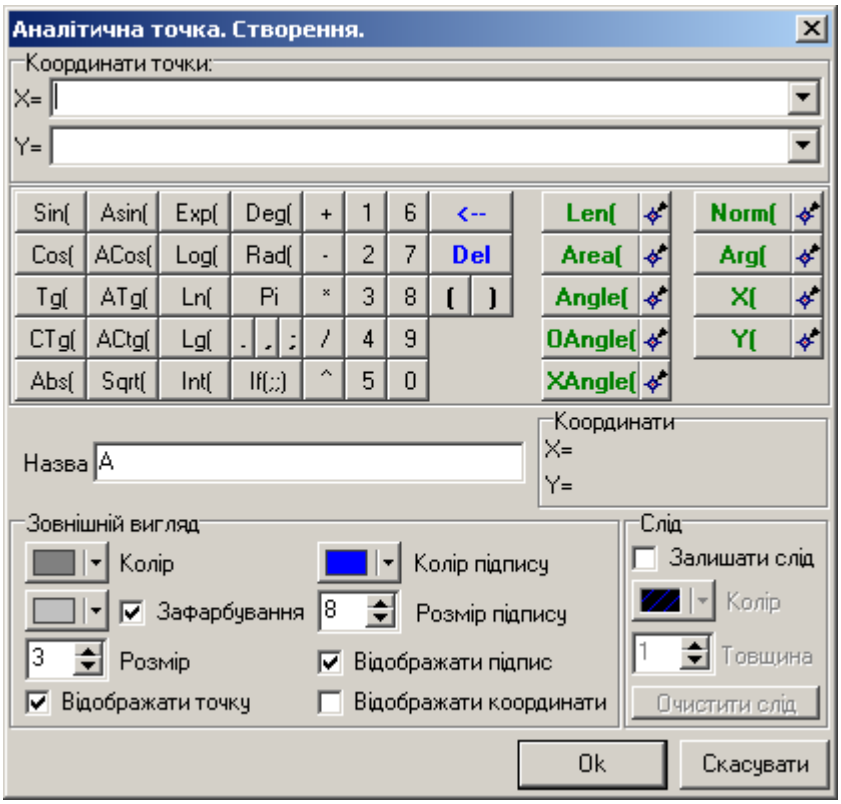


Рис. В.1. Вікно задання характеристик для створення аналітично заданої точки.

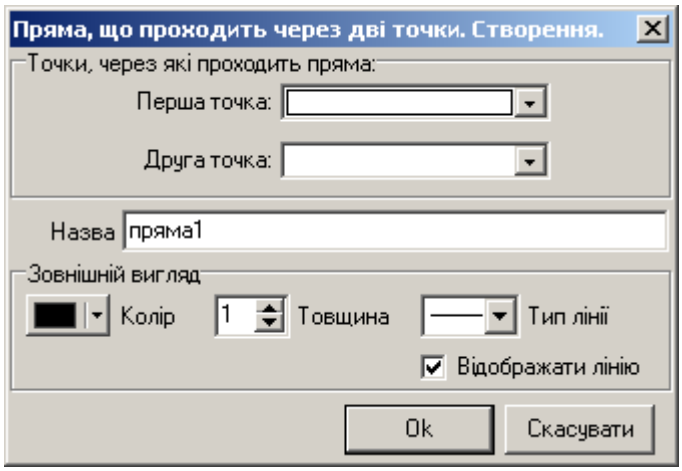


Рис. В.2. Вікно задання характеристик для створення прямої, відрізка, променя.

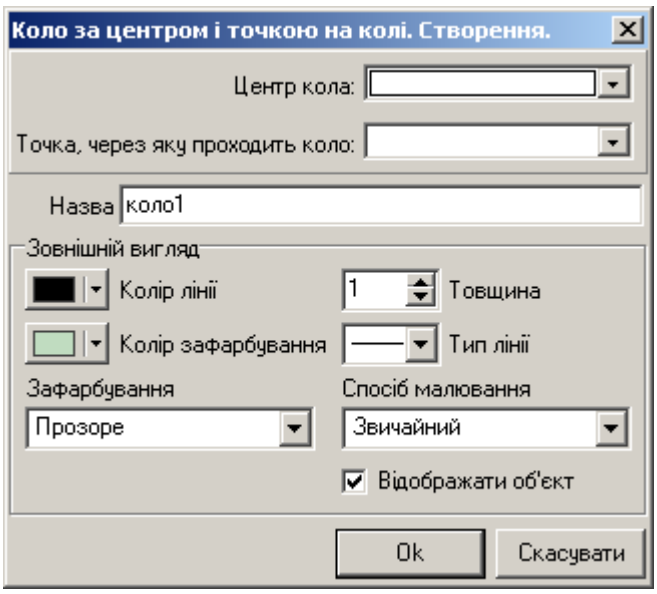


Рис. В.3. Вікно задання характеристик для створення кола за центром та точкою на колі.

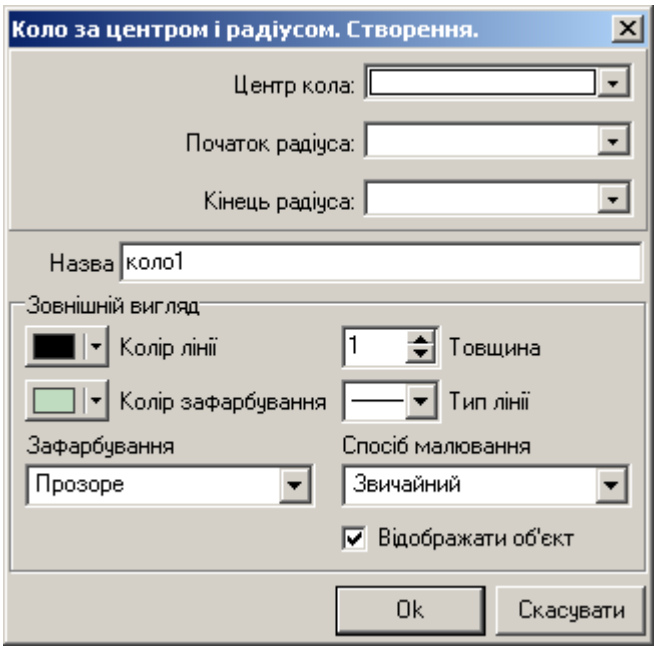


Рис. В.4. Вікно задання характеристик для створення кола за центром та точками, за якими визначається його радіус.

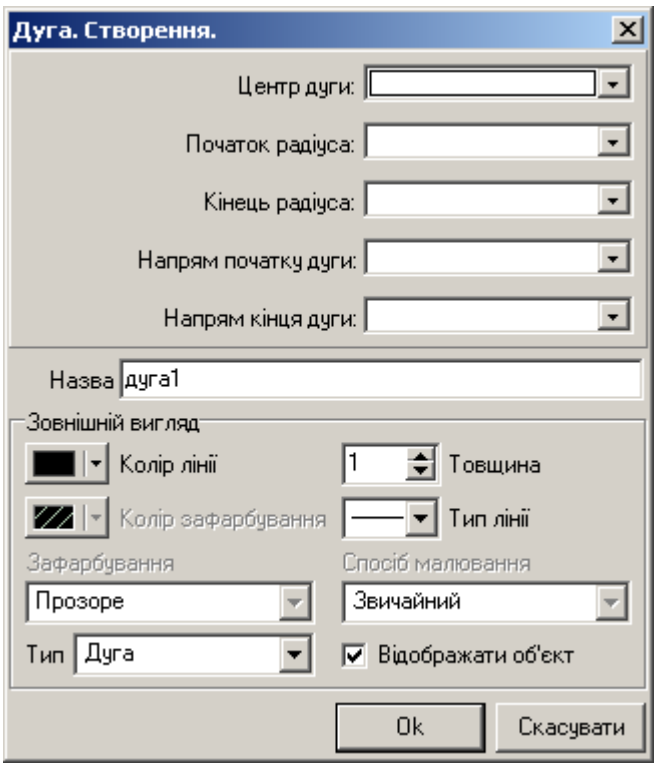


Рис. В.5. Вікно задання характеристик для створення дуги.

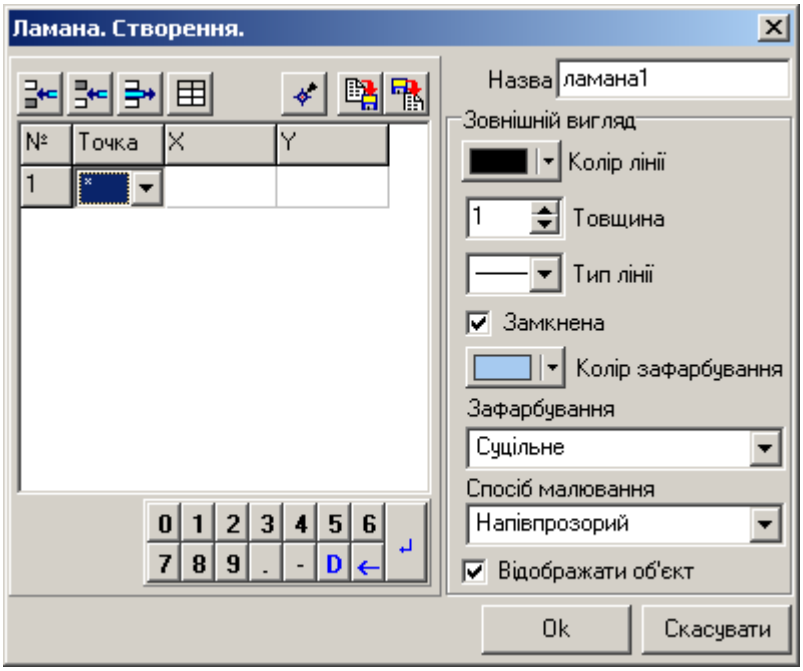


Рис. В.6. Вікно задання характеристик для створення ламаної.

Додаток Д.

Реалізація окремих методів для класів та підпрограм загального призначення ППЗ Gran2d

Методи класу TGrPlace

```
procedure TGrPlace.ScreenToAbsZ(x, y: Integer; var xx, yy:
    Extended);
begin
  XX:=(X-Fxc-FRolX)/Fzoom/FPixelsSm;
  YY:=(Fyc-Y-FRolY)/Fzoom/FPixelsSm;
end;

procedure TGrPlace.AbsToScreenZ(x, y: Extended; var xx, yy:
    Extended);
begin
  xx:=Fxc+FRolX+x*Fzoom*FPixelsSm;
  yy:=Fyc-(FRolY+y*Fzoom*FPixelsSm);
end;

procedure TGrPlace.Save(f: TIniFile; Sec: String);
begin
  f.WriteInteger(Sec, 'ColorObl', FColorObl);
  f.WriteBool   (Sec, 'PutOsi', FPutOsi);
  f.WriteInteger(Sec, 'ColorOsi', FColorOsi);
  f.WriteBool   (Sec, 'DrawGrid', FDrawGrid);
  f.WriteString (Sec, 'XNameOsi', FXNameOsi);
  f.WriteString (Sec, 'YNameOsi', FYNameOsi);
  f.WriteFloat  (Sec, 'Zoom', FZoom);
  f.WriteInteger(Sec, 'RolX', FRolX);
  f.WriteInteger(Sec, 'RolY', FRolY);
end;

procedure TGrPlace.DrawLine(x1,y1,x2,y2:Extended; Lt: Integer=0);
Var ColBounds, i: integer;
    Bounds: Array[1..2] Of TPoint;
```

```
    xx, yy: Extended;
Procedure IncBounds;
Var tx,ty:Extended;
Begin
  inc(ColBounds);
  AbsToScreenZ(xx,yy, tx,ty);
  with Bounds[ColBounds] Do
   Begin
     x:=round(tx);
     y:=round(ty);
   End;
End;

Begin
  With Canvas Do
   Begin
    ColBounds:=0;
    For i:=1 To 4 Do
     If ColBounds<2 Then
      Begin
        If X_Line_Line(x1,y1,x2,y2, AB[i].ex, AB[i].ey,
          AB[i+1].ex, AB[i+1].ey,xx,yy) Then
         If (odd(i) And (xx<=max(AB[i].ex,AB[i+1].ex)) And
          (xx>=min(AB[i].ex,AB[i+1].ex)) ) Or  (Not odd(i) And (yy<
          max(AB[i].ey,AB[i+1].ey)) And
          (yy>min(AB[i].ey,AB[i+1].ey)) )
          Then IncBounds;
      End;
    . . .
     If (Lt=0) and (colBounds>=2) Then
      Begin
        With Bounds[1] Do Moveto(x,y);
        With Bounds[2] Do lineto(x,y)
      End;
   End;
End;
```

Даний метод містить підпрограму `IncBounds`, за рахунок якої відбувається запам’ятовування координат точок перетину цієї лінії з лініями, які визначають межі робочої області.

```
procedure TGrPlace.Draw;
Var
  q, c1: Double;
  a, b, c: Double;
Procedure DrawOsi_Grid;
Begin
  With Canvas Do Begin
   If (FDrawGrid) Then
    Begin
     For j:=Trunc(AB[1].ex/q) To trunc(AB[3].ex/q) Do
      For j1:=trunc(AB[1].ey/q) DownTo trunc(AB[3].ey/q) Do
        Begin
          AbsToScreenZ(j*q,j1*q,tx,ty);
          pixels[tx,ty]:=FColorOsi;
        End;
    End;
   If FPutOsi Then
    Begin
      //побудова осей та виведення підписів на них
      . . .
    End;
  End;
End;

begin
  If FlagDrawScene Then Exit;
  DecimalSeparator:='.';
  FlagDrawScene:=True;
  Align:=alClient;
  Fxc:=Width div 2;
  Fyc:=Height div 2;
```

```
  with AB[1] do ScreenToAbsZ(0,0, ex,ey);
  with AB[2] do ScreenToAbsZ(Width,0, ex,ey);
  with AB[3] do ScreenToAbsZ(Width,Height, ex,ey);
  with AB[4] do ScreenToAbsZ(0,Height, ex,ey);
  AB[5]:=AB[1];
   Try
    Picture.Bitmap.Width:=Width;
    Picture.Bitmap.Height:=Height;
    Canvas.Brush.Color:=FColorObl;
    Picture.Bitmap.Canvas.fillRect(ClientRect);
    a:=(AB[1].ex-AB[3].ex);
    b:=(AB[1].ey-AB[3].ey);
    c:=max(a,b);
    c1:=1;
    While c<c1 Do c1:=c1/10;
    c:=c+c1;
    Q:=0.0000000000001;
    Repeat
      Q:=Q*10;
    until c/q<21;
    DrawOsi_Grid;
   Except
   End;
  FlagDrawScene:=False
end;

procedure TGrPlace.GrMouseMove(Sender: TObject; Shift:
    TShiftState; X,  Y: Integer);
begin
  ScreenToAbsZ(X,Y,FTecDX,FTecDY);
  If (ssCtrl in Shift) And (ssLeft in Shift) Then
   Begin
     FRolX:=FRolX+round((FTecDX*FPixelsSm-LastX)*FZoom);
     FRolY:=FRolY+round((FTecDY*FPixelsSm-LastY)*FZoom);
     ScreenToAbsZ(X,Y,FTecDX, FTecDY);
     Draw;
```

```
   End;
  LastX:=FTecDX*FPixelsSm;
  LastY:=FTecDY*FPixelsSm;
end;
```

Методи класу TParentObj

```
procedure TParentObj.Save(f: TIniFile; Sec: String);
Var ii: Integer;
    ss: String;
begin
  f.WriteString (Sec,'ObjType',   GetTypeName);
  f.WriteString (Sec,'Name',      FName);
  f.WriteBool   (Sec,'Visible',   FVisible);
  f.WriteBool   (Sec,'Select' ,   FSelect);
  f.WriteInteger(Sec,'RefCount',  RefCount);
  For ii:=1 To RefCount Do
   Begin
     ss:='ref'+IntTostr(ii);
     f.WriteInteger(Sec,ss,ref[ii]);
   End;
end;

procedure TParentObj.Load(f: TIniFile; Sec: String; List:
    TStrings);
Var ii: Integer;
    ss: String;
begin
  (List.Objects[List.Count-1] as TParentObj).Name:=
      f.ReadString(Sec,'Name','Noname');
  (List.Objects[List.Count-1] as TParentObj).Visible:=
      f.ReadBool(Sec,'Visible',True);
  (List.Objects[List.Count-1] as TParentObj).Select:=
      f.ReadBool(Sec,'Select', True);
  (List.Objects[List.Count-1] as TParentObj).RefCount:=
      f.ReadInteger(Sec,'RefCount',0);
```

```
   For ii:=1 To (List.Objects[List.Count-1] as
      TParentObj).RefCount Do
    Begin
        ss:='ref'+IntTostr(ii);
        (List.Objects[List.Count-1] as
          TParentObj).Ref[ii]:=f.ReadInteger(Sec,ss,-1);
    End;
end;
```

Методи класу TPointObj

```
procedure TPointObj.ShowCaption;
Var ss: String;
begin
  DLab.Visible:=((DNameShow) Or (DCoorShow)) And Visible;
  If DNameShow
   Then ss:=Name+''
   Else ss:='';
  If DCoorShow Then
   ss:=ss+' ('+SValue(FDGx)+'; '+SValue(FDGy)+')';
  DLab.GraphCaption(ss, clBlack, 8, 'Courier New');
end;
procedure TPointObj.SetDGx(const Value: Double);
Var yyy: Integer;
begin
  FDGx:= Value;
  GrPlace.AbsToScreenZ(Value, 0, Dpx, yyy);
  DLab.Left:=Dpx+10;
end;

procedure TPointObj.Draw;
Var tx, ty, sz: Integer;
begin
  DLab.Visible:=False;
  If Visible Then
   Begin
     GrPlace.Canvas.Pen.Color:=FDColor;
```

```
      GrPlace.Canvas.Pen.Style:=psSolid;
      GrPlace.Canvas.Pen.Width:=1;
      GrPlace.Canvas.Brush.Color:=FDFill;
      GrPlace.Canvas.Brush.Style:=bsSolid;
      sz:=FDSize;
      tx:=DPx;
      ty:=DPy;
      If DFillShow
       Then
        GrPlace.Canvas.ellipse(tx-sz, ty-sz, tx+sz+1, ty+sz+1)
       Else
        GrPlace.Canvas.Arc(tx-sz, ty-sz, tx+sz+1, ty+sz+1, tx, ty-
         sz, tx, ty-sz);
  End;
end;
```

Методи класу TLineObj

```
procedure TLineObj.SetPropLine2Dot(xx1, yy1, xx2, yy2: Extended);
begin
  Gx1:=xx1;
  Gy1:=yy1;
  GrPlace.AbsToScreenZ(xx1, yy1, Px1, Py1);
  Gx2:=xx2;
  Gy2:=yy2;
  GrPlace.AbsToScreenZ(xx2, yy2, Px2, Py2);
  Ax:=Gy2-Gy1;
  By:=Gx1-Gx2;
  Cz:=Gy1*(Gx2-Gx1)-Gx1*(Gy2-Gy1);
end;

procedure TLineObj.Draw;
begin
  If Visible Then
   Begin
     GrPlace.Canvas.Pen.Color:=LColor;
     GrPlace.Canvas.Pen.Style:=PStyles[LStyle];
```

```
      GrPlace.Canvas.Pen.Width:=LSize;
      GrPlace.Canvas.Brush.Color:=c_ColorObl;
      GrPlace.Canvas.Brush.Style:=bsSolid;
      SetPropLine2Dot(Gx1, Gy1, Gx2, Gy2);
      GrPlace.DrawLine(Gx1, Gy1, Gx2, Gy2, 0);
    End;
end;

procedure TLineObj.ShowSett(M: TStrings; Move: Boolean=False);
Var sa: String;
begin
  M.Clear;
  sa:=ShowInfo;
  M.Add(sa);
  If TypeObj=shLineVidriz
   Then M.Add(Format('Довжина відрізку:
      %s',[SValue(Len_Dot_Dot(Gx1,Gy1, Gx2,Gy2))]));
  sa:='';
  If Ax<>0 Then sa:=SValue(Ax)+'x';
  If By<>0 Then
   Begin
     If By<0
      Then sa:=sa+SValue(By)+'y'
      Else sa:=sa+'+'+SValue(By)+'y';
   End;
  If Cz<>0 Then
   Begin
     If Cz<0 Then sa:=sa+SValue(Cz)
             Else sa:=sa+'+'+SValue(Cz);
   End;
  M.Add(Format('Рівняння прямої: %s=0',[sa]));
end;
```

де функція `len_Dot_Dot` призначена для обчислення відстані між двома точками, заданими своїми координатами.

Методи класу TTextObj

```
procedure TTextObj.LabelMouseDown(Sender: TObject; Button:
    TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  If Button<>mbLeft
   Then FlagMove:=False
   Else Begin
    FlagMove:=True;
    XOld:=X;
    YOld:=Y;
   End
end;

procedure TTextObj.LabelMouseMove(Sender: TObject; Shift:
    TShiftState; X, Y: Integer);
begin
  If (FlagMove) Then
   Begin
    Left:=Left+X-XOld;
    Top:=Top +Y-YOld;
   End;
end;

procedure TTextObj.LabelMouseUp(Sender: TObject; Button:
    TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  FlagMove:=False;
end;
```

Методи класу TTextPoint

```
procedure TTextPoint.LabelMouseMove(Sender: TObject; Shift:
      TShiftState; X, Y: Integer);
Var w, h: Double;
begin
```

```
  w:=Width/2;
  h:=Height/2;
  If sqr((Left+X-XOld-(Dx-w))/(15+w))+sqr((Top+Y-YOld-(Dy-
      h))/(15+h))<1
   Then inherited;
  FWith:=Left-Dx;
  FHeight:=Top-Dy;
end;
```

Підпрограми загального призначення

```
function X_Line_Line(x11,y11, x12,y12, x21,y21, x22,y22: Extended;
    var x,y: Extended): Boolean;
Var x1,y1,x2,y2,a1,b1,c1,a2,b2,c2: Extended;
begin
  Result:=True;
  x1:=x12-x11; y1:=y12-y11;
  x2:=x22-x21; y2:=y22-y21;
  a1:=y1; b1:=-x1; c1:=y11*x1-x11*y1;
  a2:=y2; b2:=-x2; c2:=y21*x2-x21*y2;
  If (a1*b2-a2*b1=0) Or (a1*b2-a2*b1=0) Then
   Begin
     Result:=False;
     Exit;
   End;
  x:= (b1*c2-b2*c1)/(a1*b2-a2*b1);
  y:=-(a1*c2-a2*c1)/(a1*b2-a2*b1);
end;

Procedure ReCalcActObj(n: Integer);
Var j: Integer;
Begin
  With FMain.ListObj.Items Do
   For j:=1 To (Objects[n] as TParentObj).RefCount Do
    With (Objects[n] as TParentObj) Do
     Begin
```

```
      If ((Objects[ref[j]] as TParentObj).ReCalc=False) And
         ((Objects[ref[j]] as TParentObj).RefCount>0)
       Then ReCalcActObj(ref[j]);
      (Objects[ref[j]] as TParentObj).ReCalc:=True;
    End;
  With FMain.ListObj.Items Do
   Case (Objects[n] as TParentObj).TypeObj Of
     shDot: Begin
        (Objects[n] as TPointObj).DGx:=
        (Objects[n] as TPointObj).DGx;
        (Objects[n] as TPointObj).DGy:=
        (Objects[n] as TPointObj).DGy;
     End;
     shLineVidriz..shLine : Begin
        (Objects[n] as TLineObj).SetPropLine2Dot(
          (FMain.ListObj.Items.Objects[(Objects[n] as
          TParentObj).Ref[1]] as TPointObj).DGx,
          (FMain.ListObj.Items.Objects[(Objects[n] as
          TParentObj).Ref[1]] as
          TPointObj).DGy,(FMain.ListObj.Items.Objects[(Objects[n]
          as TParentObj).Ref[2]] as
          TPointObj).DGx,(FMain.ListObj.Items.Objects[(Objects[n]
          as TParentObj).Ref[2]] as TPointObj).DGy);
      End;
      . . .
   End;
End;

Procedure ReCalcObj;
Var i: Integer;
Begin
  DecimalSeparator:='.';
  With FMain.ListObj.Items Do
  Begin
    For i:=0 To Count-1 Do
     (Objects[i] as TParentObj).ReCalc:=False;
```

```
    For i:=0 To Count-1 Do
     Begin
     Try
      If ((Objects[i] as TParentObj).ReCalc=False) And
        ((Objects[i] as TParentObj).RefCount>0)
       Then ReCalcActObj(i);
      (Objects[i] as TParentObj).ReCalc:=True;
      If (Objects[i] as TParentObj).TypeObj=shDot Then
       Begin
        (Objects[i] as TPointObj).DGx:=(Objects[i] as
          TPointObj).DGx;
        (Objects[i] as TPointObj).DGy:= (Objects[i] as
          TPointObj).DGy;
       End;
       Except
       End;
      End;
  End;
  If (FMain.ListObj.ItemIndex<>-1)
   Then FMain.ListObjClick(Application);
End;
```

Додаток Е.

Реалізація окремих методів для класів та підпрограм загального призначення ППЗ Numet

Методи класу TFunc

```
procedure TFunc.ChangeFunc(SF: String; var Chng:Boolean);
var   Err:Boolean;
      NTx, NSt: String;
      NArr: RealArray;
begin
  Err:=False;
  NTx:=SF;
  NSt:=UpperCase(NTx);
  Convertor(NSt, NArr, Err);
  if Err then
      Chng:=False
  else
    begin
      Tx:=NTx; St:=NSt; Arr:=NArr;
      Chng:=True;
    end;
end;

function TFunc.Eval(ArgX, ArgY: Extended): Extended;
  var stack:array[1..10] of extended;
      y,z,w:extended;
      i,j,top,xx,integ1,integ2:integer;
begin
  Top:=0;
  i:=1;
  try
  while i<=Length(St) do
    begin
      case St[i] of
       #0..chr(numofconst):begin
```

```
        stack[top+1]:=Arr[Byte(St[i])];inc(top)
                    end;
 #30:begin
       stack[top+1]:=ArgX;inc(top)
     end;
 #29:begin
       stack[top+1]:=ArgY;inc(top)
     end;
#255:begin
       y:=stack[top]; stack[top]:=stack[top-1]; stack[top-
        1]:=y
     end;
 '"':begin
       stack[top-1]:=stack[top-1]+stack[top];dec(top)
     end;
 '#':begin
       stack[top-1]:=stack[top-1]-stack[top];dec(top)
     end;
 '$':begin
       stack[top-1]:=stack[top-1]*stack[top];dec(top)
     end;
 '%':begin
       if stack[top]<>0 then stack[top-1]:=stack[top-
        1]/stack[top] else begin stack[top]:=AllErr; Break;
        end;
       dec(top)
     end;
 '&':begin
       if stack[top-1]=0 then
         dec(top)
       else
         begin
           if stack[top]=0 then
             begin
               stack[top-1]:=1;
```

```
        dec(top)
      end
    else
      begin
        y:=ln(abs(stack[top-1]))*stack[top];
        if (stack[top]>0) and (stack[top]<MaxInt) and
(int(stack[top])=stack[top]) then
          begin
            z:=1;integ1:=round(stack[top]);
            while integ1>0 do
              begin
                integ2:=integ1 div 2;
                if (integ2+integ2)<integ1 then
z:=z*stack[top-1];
                integ1:=integ2;
                stack[top-1]:=stack[top-1]*stack[top-
1];
              end;
            stack[top-1]:=z;
            dec(top)
          end
        else
          begin
            y:=exp(y);
            z:=abs(stack[top]);
            xx:=0;
            if (stack[top-1]<0) then
            begin
              if (z<=1) and (z>=(1/1e+3)) and
(int(1/z)=1/z) and ((round(1/z) mod 2)=1) then
              stack[top-1]:=sign(stack[top-1])*y
              else
              begin
              if (z>=1) and (int(z)=z) then
              begin
                w:=1;integ1:=round(z);
```

```
                        while integ1>0 do
                        begin
                          integ2:=integ1 div 2;
                          if (integ2+integ2)<integ1 then
        w:=w*stack[top-1];
                          integ1:=integ2;
                          stack[top-1]:=stack[top-
        1]*stack[top-1];
                        end;
                        stack[top-1]:=1/w;
                      end
                      else
                      begin
                        stack[top]:=AllErr;
                        Break;
                      end
                      end
                    end
                    else
                      stack[top-1]:=y;
                    dec(top);
                  end;
              end;
          end;
      end;
'H':stack[top]:=arctan(stack[top]);
'B':stack[top]:=cos(stack[top]);
'J':stack[top]:=exp(stack[top]);
'L':if stack[top]>0 then stack[top]:=ln(stack[top]) else
        begin stack[top]:=AllErr; Break; end;
'K':if stack[top]>0 then stack[top]:=ln(stack[top])/Ln10
        else begin stack[top]:=AllErr; Break; end;
'M':stack[top]:=abs(stack[top]);
'N':if stack[top]>=0 then stack[top]:=sqrt(stack[top]) else
        begin stack[top]:=AllErr; Break; end;
'A':stack[top]:=sin(stack[top]);
```

```
        'C':stack[top]:=sin(stack[top])/cos(stack[top]);
        'D':stack[top]:=cos(stack[top])/sin(stack[top]);
        'F':if abs(stack[top])<1 then
              stack[top]:=arctan(stack[top]/sqrt(1-
              stack[top]*stack[top])) else begin
              stack[top]:=AllErr; Break; end;
        'G':if abs(stack[top])<1 then stack[top]:=pi/2-
              arctan(stack[top]/sqrt(1-stack[top]*stack[top])) else
              begin stack[top]:=AllErr; Break; end;
        'I':stack[top]:=pi/2-arctan(stack[top]);
        'R':if stack[top]>=0 then stack[top]:=int(stack[top]) else
              stack[top]:=int(stack[top])-1;
      end;
      i:=i+1;
    end;
  Eval:=stack[top];
  except
    Eval:=AllErr;
  end;
end;
```

Методи класу TNumerMet

```
constructor TNumerMet.create;
begin
  inherited create;
  TypeFunc:=tfNone;
  PEps:=0.001;
end;

procedure TNumerMet.Load(f : TIniFile; Sec : String; List:
        TStrings);
Var TypeName : String;
    ii, jj : Integer;
Begin
  TypeName:=f.ReadString(Sec,'ObjType','');
```

```
  With List Do
   Begin
     if TypeName='Normal'
      Then Begin
        AddObject('Y(X)='+f.ReadString(Sec, 'Func', 'X'),
          TMetFun.Create);
        (Objects[Count-1] as TMetFun).Xa:= f.ReadFloat(Sec,'A',0);
        (Objects[Count-1] as TMetFun).Xb:= f.ReadFloat(Sec,'B',0);
        (Objects[Count-1] as TMetFun).Func:=
          f.ReadString(Sec,'Func','X');
        (Objects[Count-1] as TMetFun).FuncType:=Normal;
      End;
    if TypeName='NormalY'
      Then Begin
        AddObject('F(X,Y)='+f.ReadString(Sec,'Func','X'),
          TMetDiff.Create);
        (Objects[Count-1] as TMetDiff).Xa:=
          f.ReadFloat(Sec,'A',0);
        (Objects[Count-1] as TMetDiff).Xb:=
          f.ReadFloat(Sec,'B',0);
        (Objects[Count-1] as TMetDiff).Func:=
          f.ReadString(Sec,'Func','X');
        (Objects[Count-1] as TMetDiff).FuncType:=NormalY;
           End;
    . . .
    End;
End;
```

Методи класу TMetFuncs

```
procedure TMetFuncs.InputInfo(M: TStrings);
begin
  If GetTypeName='Normal'
   Then M.Lines.Add('Y(X)='+Func);
  If GetTypeName='NormalY'
   Then M.Lines.Add('F(X,Y)='+Func);
```

```
  M.Lines.Add('A='+Svalue(Xa));
  M.Lines.Add('B='+Svalue(Xb));
end;

procedure TMetFuncs.Save(f: TIniFile; Sec: String);
begin
  f.WriteString(Sec,'ObjType',GetTypeName);
  f.WriteString(Sec,'Func',Func);
  f.WriteString(Sec,'A',Svalue(Xa));
  f.WriteString(Sec,'B',Svalue(Xb));
end;
```

Методи класу TMetSlr

```
function TMetSlr.Sumistnist: TypeSumist;
Var TmpSp : TypeSumist;
    TmpKoefSp, KoefSp : Real;
    kolSp, ii : Integer;
    Tmp : Boolean;
    i, j : Integer;
begin
  TmpSp:=FRozv;
  Tmp:=False;
  KoefSp:=0;
  For i:=kolrow DownTo 2 Do
   For j:=1 DownTo i-1 Do
    Begin
      If TmpSp=FRozv
       Then ii:=1
       Else ii:=kolcol+1;
      kolsp:=0;
      While (ii<kolcol) Do
       Begin
        While (ii<kolcol) And (mt[i,ii]=mt[j,ii]) And (mt[i,ii]=0)
         Do Begin
          ii:=ii+1;
```

```
          kolsp:=kolsp+1;
         End;
        While (ii<kolcol) And ((mt[i,ii]<>0) Or (mt[j,ii]<>0))
         Do Begin
          If mt[j,ii]=0
           Then TmpKoefSp:=0
           Else TmpKoefSp:=mt[i,ii]/mt[j,ii];
          If Not Tmp
           Then Begin
            KoefSp:=TmpKoefSp;
            Tmp:=True;
           End;
          If KoefSp=TmpKoefSp
           Then KolSp:=KolSp+1
           Else ii:=kolcol;
          ii:=ii+1;
         End;
       End;
      If KolSp=kolrow
       Then Begin
        TmpSp:=FRang;
        If mt[j,kolcol]=0
         Then TmpKoefSp:=0
         Else TmpKoefSp:=mt[i,kolcol]/mt[j,kolcol];
        If KoefSp=TmpKoefSp
         Then TmpSp:=FLZaleg;
        If (KolRow=2) And (((mt[1,1]=0) And (mt[2,2]=0)) Or
        ((mt[1,2]=0) And (mt[2,1]=0)))
         Then TmpSp:=FRozv;
       End;
    End;
  Sumistnist:=TmpSp;
end;
```

Методи класу TGenHtm

```
Procedure TGenHtm.OutputWindow(WebOut : TwebBrowser);
Var _URL, Flags, TargetName, PostData, Headers: Olevariant;
    app : TApplication;
    StDir : String;
Begin
  StDir:=ExtractFilePath(App.EXEName);
  _URL:=StDir+'Myweb.html';
  HtmlList.SaveToFile(_URL);
  Flags:=0; TargetName:=0; Postdata:=0; Headers:=0;
  WeBOut.Navigate2(_URL, Flags, TargetName, PostData, Headers);
End;

Procedure TGeHtm.GenTabHtm(StGH : String);
Var p1, p2, ij : Byte;
    std, stpar, st, stalig : String;
    stcol, strow : String;
    EndRow : Boolean;
Begin
  If (Stgh[1]='\') And (Stgh[2]='n') And (Stgh[3]='\')
   Then  Begin
           OutputList('<table style=''border:none''
          class=MsoNormalTable border=1 cellspacing=0
          cellpadding=0>');
           Delete(Stgh,1,3);
         End
   Else  OutputList('<table class=MsoNormalTable border=1
          cellspacing=0 cellpadding=0>');
  While Length(StGh)>0 Do
   Begin
     OutputList(' <tr>');
     Repeat
        p1:=pos('\',stGH);
        std:=copy(StGh,1,p1-1);
        Delete(StGh,1,p1);
        p2:=pos('\',stGH);
        stpar:=copy(StGh,1,p2-1);
```

```
Delete(StGh,1,p2);
st:=‘  <td’;
If pos(‘"‘,Stpar)<>0
 Then Begin
        p1:=pos(‘"‘,Stpar);
        strow:=Copy(Stpar,1,p1-1);
        Delete(Stpar,1,p1);
        p1:=pos(‘"‘,Stpar);
        stcol:=Copy(Stpar,1,p1-1);
        Delete(Stpar,1,p1);
        If strow<>‘1’ Then st:=st+’ rowspan=‘+strow;
        If stcol<>‘1’ Then st:=st+’ colspan=‘+stcol;
      End
 Else Begin
        If stpar[1]<>‘1’ Then st:=st+’ rowspan=‘+stpar[1];
        If stpar[2]<>‘1’ Then st:=st+’ colspan=‘+stpar[2];
        Delete(Stpar,1,2);
      End;
If stpar[Length(stpar)]=‘|’
 Then Begin
        EndRow:=True;
        Delete(stpar,Length(stpar),1);
      End
 Else EndRow:=False;
If Length(stpar)>1
 Then Begin
        st:=st+’ style=‘‘‘;
        If stpar[2] in [‘q’,’w’,’e’]
         Then Case stpar[2] Of
                ‘q’ : st:=st+’background: #FFCCCC;’;
                ‘w’ : st:=st+’background: #80FF80;’;
                ‘e’ : st:=st+’background: #C0C0C0;’;
              End;
        For ij:=2 To Length(stpar) Do
         Case stpar[ij] Of
          ‘L’ : st:=st+’ border-left:none;’;
```

```
                    'T' : st:=st+' border-top:none;';
                    'R' : st:=st+' border-right:none;';
                    'B' : st:=st+' border-bottom:none;';
                   End;
                  st:=st+' windowtext 1.0pt''>';
               End
          Else st:=st+'>';
         OutputList(st);
         Case stpar[1] Of
           'l' : std:='    <div
           align="left"> '+std+' </div></td>';
           'c' : std:='
           <center> '+std+' </center></td>';
           'r' : std:='    <div
           align="right"> '+std+' </div></td>';
         End;
         OutputList(std);
      Until EndRow;
      OutputList(' </tr>');
   End;
  OutputList('</table>');
End;
```

Підпрограми загального призначення

```
function AfterPoint(Data: Extended) : Integer;
Var apk : integer;
    s : String;
Begin
  If Data<0 Then Data:=-1*Data;
  Data:=Frac(Data);
  s:=Svalue(Data);
  apk:=Length(s)-2;
  If apk=-1 Then apk:=0;
  AfterPoint:=apk;
End;
```

Додаток Ж.

Анкети щодо готовності добору готового чи створення власного ППЗ

Додаток Ж.1 Анкета для вчителів

1. На вашу думку до ППЗ належать (можна обрати декілька варіантів):

	Комп'ютерні програми, призначені для комп'ютерної підтримки навчання.
	Електронні підручники та довідники.
	Гіпертекстові електронні підручники та довідники.
	Комп'ютерні системи тестування.
	Презентації.
	Будь який навчальний матеріал, що може бути представлений за рахунок використання комп'ютера.

2. Оцініть свій рівень знань з точки зору щодо можливого застосування ППЗ в своїй педагогічній діяльності (достатній, середній, низький)?

3. Чи використовує те ви ППЗ в своїй педагогічній діяльності (завжди, досить часто, інколи, ніколи)?

4. Якими мовами програмування володієте?

5. Чи були намагання створити власний ППЗ для його використання в навчальному процесі (так / ні)?

6. Чи створювали Ви власні ППЗ для їх використання в навчальному процесі (так / ні)?

7. Як Ви оцінюєте рівень своїх знань та вмінь для створення власного ППЗ (можна обрати декілька варіантів):

	Недостатній рівень знань з алгоритмізації та програмування.
	Недостатній рівень знань щодо роботи в середовищах програмування для створення ППЗ з графічним інтерфейсом.
	Недостатній рівень знань підходів та технологій розробки ППЗ.
	Недостатній рівень знань з побудови ієрархій класів ППЗ для відповідної предметної галузі.
	Цілком достатній рівень знань для розробки власного ППЗ.

8. Пронумеруйте характеристики згідно їх важливості для ППЗ (1 – найбільш важлива, 8 – найменш важлива):

	Зовнішній вигляд ППЗ.
	Наявність сучасних засобів візуалізації (наприклад, засобів комп'ютерної графіки, технології мультимедіа) об'єктів, процесів явищ.
	Забезпечення науковості змісту ППЗ.
	Враховування своєрідності і особливості конкретного навчального предмета, специфіки відповідної науки, її понятійного апарату.

	Врахування вікових і індивідуальних особливостей учнів.
	Функціональна повнота ППЗ.
	Стійкість до помилкових, некоректних та несанкціонованих дій користувача.
	Вільно поширюваність ППЗ.
	Україномовний інтерфейс.

9. Чи зверталися до Вас інші вчителі з проханням добрати доцільний ППЗ (якщо так то вчителі яких предметів)?

10. Чи зверталися до Вас інші вчителі з проханням розробити ППЗ (якщо так то вчителі яких предметів)?

Відповісти на наступні питання тільки, якщо в вашій педагогічній діяльності Ви використовуєте ППЗ.

11 Вкажіть основні недоліки використовуваних Вами ППЗ.

12. Як ви використовуєте ППЗ в своїй професійній діяльності (можна обрати декілька варіантів)?

	Для підготовки наочних матеріалів.
	Для особистої перевірки навчальних завдань.
	Для пояснення нового матеріалу.
	Для закріплення навчального матеріалу.
	Для контролю практичних навичок учнів.
	Для побудови комп’ютерних моделей математичних об’єктів для дослідження їх властивостей, пошуку закономірностей, формування і підтвердження (спростування) гіпотез.
	Для побудови автоматичних розв’язників математичних задач.

Додаток Ж.2 Анкета для студентів

1. На вашу думку до ППЗ належать (можна обрати декілька варіантів):

	Комп’ютерні програми, призначені для комп’ютерної підтримки навчання.
	Електронні підручники та довідники.
	Гіпертекстові електронні підручники та довідники.
	Комп’ютерні системи тестування.
	Презентації.
	Будь який навчальний матеріал, що може бути представлений за рахунок використання комп’ютера.

2. Чи розглядалося, на окремих вузівських дисциплінах, можливе застосування ППЗ в навчальному процесі?

3. Оцініть свій рівень знань з точки зору щодо можливого застосування ППЗ в своїй майбутній педагогічній діяльності (достатній, середній, низький)?

4. Якими мовами програмування володієте?

5. Чи намагалися Ви дібрати ППЗ для підтримки викладання окремих шкільних предметів?

6. Вкажіть основні недоліки розглядуваних Вами ППЗ.

7. Як Ви оцінюєте рівень своїх знань та вмінь для створення власного ППЗ (можна обрати декілька варіантів):

	Недостатній рівень знань з алгоритмізації та програмування.
	Недостатній рівень знань щодо роботи в середовищах програмування для створення ППЗ з графічним інтерфейсом.
	Недостатній рівень знань підходів та технологій розробки ППЗ.
	Недостатній рівень знань з побудови ієрархій класів ППЗ для відповідної предметної галузі.
	Цілком достатній рівень знань для розробки власного ППЗ.

8. Пронумеруйте характеристики згідно їх важливості для ППЗ (1 – найбільш важлива, 8 – найменш важлива):

	Зовнішній вигляд ППЗ.
	Наявність сучасних засобів візуалізації (наприклад, засобів комп’ютерної графіки, технології мультимедіа) об’єктів, процесів явищ.
	Забезпечення науковості змісту ППЗ.
	Враховування своєрідності і особливості конкретного навчального предмета, специфіки відповідної науки, її понятійного апарату.
	Врахування вікових і індивідуальних особливостей учнів.
	Функціональна повнота ППЗ.
	Стійкість до помилкових, некоректних та несанкціонованих дій користувача.
	Вільно поширюваність ППЗ.
	Україномовний інтерфейс.

9. Як на Вашу думку може використовуватися ППЗ в навяальному процесі (можна обрати декілька варіантів)?

	Для підготовки наочних матеріалів.
	Для особистої перевірки навчальних завдань.
	Для пояснення нового матеріалу.
	Для закріплення навчального матеріалу.
	Для контролю практичних навичок учнів.
	Для побудови комп’ютерних моделей математичних об’єктів для дослідження їх властивостей, пошуку закономірностей, формування і підтвердження (спростування) гіпотез.
	Для побудови автоматичних розв’язників математичних задач.

Додаток З.

Орієнтований перелік тестових завдань для попереднього тестування при формуванні контрольної та експериментальної груп

Запитання №1

Послідовність літер і цифр, що починається з літери називається:

Відмітьте вірні варіанти:

	оператор
х	ідентифікатор
	селектор
	слово
	алфавіт

Запитання №2

В блоці опису Pascal-програма може містити?

Відмітьте вірні варіанти:

	опис коментарів
х	опис зовнішніх модулів
х	опис констант
х	опис змінних
х	опис типів

Запитання №3

Правильно написана підпрограма спілкується з основною програмою через

Відмітьте вірні варіанти:

х	параметри
	константи
	модулі
	процедури
	функції

Запитання №4

Під рекурсією розуміють:

Відмітьте вірні варіанти:

	виклик процедури із тіла функції.
х	виклик процедури (функції) з тіла цієї ж самої процедури (функції).
	виклик процедури (функції) із головної програми.
	виклик процедури (функціх) із тіла іншої процедури (функції).
	виклик функції із тіла процедури.

Запитання №5

Об’єктно-орієнтовне програмування - це

Відмітьте вірні варіанти:

	програмування в середовищі Lazarus.
	методологія проектування, що з'єднує в собі процес об'єктної декомпозиції й прийоми подання логічної й фізичної, а також статичної й динамічної моделей проектованої системи.
x	методологія програмування, основана на представленні програми у вигляді сукупності об’єктів, кожен з яких є екземпляром певного класу (класи утворюють ієрархію наслідування) і має інтерфейс у вигляді набору методів для взаємодії один з одним.
	однин з видів мов програмування.

Запитання №6

До основних концепцій (принципів) ООП відносяться:

Відмітьте вірні варіанти:

	Перезавантаження
	Агрегація
x	Наслідування
x	Інкапсуляція
x	Поліморфізм

Запитання №7

Об’єкт - це

Відмітьте вірні варіанти:

	поняття, абстракція чи будь-яка річ, яка володіє станом, поведінкою і ідентичністю.
x	поняття, абстракція чи будь-яка річ з чітко окресленими межами, яка має смисл в контексті розглядуваної прикладної проблеми.
	поняття, абстракція чи будь-яка річ з чітко окресленими межами.
x	екземпляр класу чи змінна типу даних, визначеного цим класом.

Запитання №8

Клас - це

Відмітьте вірні варіанти:

	тип даних описаний в блоці описів програми.
x	визначений користувачем тип даних, який має стан (його представлення: поля чи властивості), ряд операцій (його поведінка: методи) та подій, на які він може реагувати.
x	деяка множина об’єктів, які мають спільну структуру і спільну поведінку.

Запитання №9

Абстрактний клас - це

Відмітьте вірні варіанти:

	клас, в якому відсутні поля.
	клас, в якому існують абстрактні методи.
	клас TObject.
x	клас, який не може мати екземплярів.

Запитання №10

Інкапсуляція це

Відмітьте вірні варіанти:

	створення класу з використанням властивостей
x	механізм об’єднання даних та коду, який маніпулює цими даними, а також захисту і того і іншого від зовнішнього втручання, або неправильного використання
	спеціальний механізм класів, який регулює доступ до них
	механізм мови ООП, який дозволяє використовувати властивості вже існуючого класу при описі нового класу
	залежність між складеним класом об’єктів і класами, які представляють компоненти цих об’єктів

Запитання №11

Інкапсуляція може бути заснована на

Відмітьте вірні варіанти:

x	модулях
	додатках
	полях
x	класах
	властивостях

Запитання №12

Зони видимості атрибутів об'єктів можуть починатися з директиви

Відмітьте вірні варіанти:

x	published
x	protected
x	private
x	public

Запитання №13

Обробка виключних ситуацій виконується за допомогою команд

Відмітьте вірні варіанти:

x	finally
x	try
x	except
	raise

Запитання №14

Згідно правила сумісності типів

Відмітьте вірні варіанти:

x	завжди можна використовувати об’єкт класу нащадка, коли очікується об’єкт класу предка, але не можна використовувати об’єкт класу предка коли очікується об’єкт класу нащадка.
	використовувати об’єкт класу нащадка, коли очікується об’єкт класу предка можна лише з явним приведенням класів.
	завжди можна використовувати об’єкт класу нащадка, коли очікується об’єкт класу предка, та об’єкт класу предка коли очікується об’єкт класу нащадка.
	завжди можна використовувати об’єкт класу предка, коли очікується об’єкт класу нащадка, але не можна використовувати об’єкт класу нащадка коли очікується об’єкт класу предка.

Запитання №15

При оголошенні методів можуть бути використані такі ключі

Відмітьте вірні варіанти:

x	override
x	abstract
x	virtual
x	dynamic
x	overload

Запитання №16

Перезавантаження методів

Відмітьте вірні варіанти:

	відбувається при описі методу з вказанням ключа override.
x	відбувається при описі методу з вказанням ключа overload.
	надає можливість використовувати різні імена для методів з однаковою сигнатурою.
x	надає можливість використовувати однакові імена для методів з різною сигнатурою.
x	надає можливість використовувати однакові імена для методів з однаковою сигнатурою але різним типом результату.

Запитання №17

Перевизначення та підміна методу може використовуватися для

Відмітьте вірні варіанти:

	використання можливостей поліморфізму
x	оптимізації методу
x	розширення можливостей методу
x	обмеження можливостей методу

Запитання №18

Модуль даних може містити блоки визначені наступними описами:

Відмітьте вірні варіанти:

x	finalization
x	implementation
x	initialization
x	interface

Запитання №19

В частині реалізації

Відмітьте вірні варіанти:

	вказуються оператори, які мають бути виконані до передачі управлінню основній програмі і найчастіше використовуються для підготовки її роботи
x	можуть бути оголошені локальні для модуля елементи - додаткові типи, константи, змінні і описані підпрограми, які будуть доступні лише з даного модуля
	вказуються оператори, які мають бути виконані після завершення роботи основної програми
	міститься оголошення всіх глобальних елементів модуля (типів, констант, змінних і підпрограм), які мають бути доступними основній програмі та іншим модулям (до яких буде підєднано даний)

Запитання №20

Властивість

Відмітьте вірні варіанти:

	це механізм об’єднання даних та коду, який маніпулює цими даними, а також захисту і того і іншого від зовнішнього втручання, або невірного використання
	це атрибут описаний в розділі public
	є однією з основних характеристик класу
	є однією з характеристик класу
x	це спеціальний механізм класів, який регулює доступ до них

Запитання №21

Які з наведених описів властивостей є правильним

Відмітьте вірні варіанти:

x	property X : Real write FX;
x	property X : Real read FX write SetX;
x	property X : Real read GetX;
x	property X : Real write FX;
x	property X : Real read FX;

Запитання №22

Вкажіть послідовну ієрархію структури класів (чим вище клас в ієрархіє тим менше число)

Проставте номери в правильній послідовності:

5	TWinControl
4	TControl
1	TObject
3	TComponent
2	TPersistent

Запитання №23

Компонентні класи можна поділити на такі групи є:

Відмітьте вірні варіанти:

	Об'єктні елементи управління
x	Графічні елементи управління
x	Віконні елементи управління
x	Елементи управління
x	Невізуальні компоненти

Запитання №24

Для зчитування даних з файлу ініціалізація пов'язаного з класом TIniFile необхідно знати

Відмітьте вірні варіанти:

x	тип значення параметру що буде зчитуватися
	номер рядку з якого буде проводитися зчитування
x	назву секції з якої буде проводитися зчитування
x	ім'я параметру значення якого буде зчитуватися

Запитання №25

Файли ініціалізації які пов'язані з класом TIniFile являють собою

Відмітьте вірні варіанти:

x	звичайний текстовий файл
x	файл вміст якого логічно згруповано в секції
	спеціальний бінарний файл

Додаток И.
Орієнтований перелік тестових запитань для підсумкового контролю з курсу “Технології програмування та створення ППЗ”

Запитання №1

Серед технологій програмування можна визначити:

Відмітьте вірні варіанти:

x	модульну
	імперативну
x	об’єктно-орієнтовну
x	структурну
	логічну

Запитання №2

До основних парадигм програмування відносяться:

Відмітьте вірні варіанти:

x	декларативне програмування
x	імперативне програмування
	модульне програмування
x	об’єктно-орієнтовне програмування
	структурне програмування

Запитання №3

Технологія програмування - це

Відмітьте вірні варіанти:

	сукупність ідей та понять, які визначають стиль написання програми
x	система методів, способів і прийомів розробки і налагодження програм
	мультипарадигменність мов програмування
	система правил визначення поводження окремих мовних конструкцій

Запитання №4

Парадигми програмування - це

Відмітьте вірні варіанти:

	система методів, способів і прийомів розробки і налагодження програм
x	сукупність ідей та понять, які визначають стиль написання програми
	спосіб бачення світу
	набір теорій, стандартів та методів, які спільно представляють собою спосіб організації наукового знання

Запитання №5

Парадигма програмування, в якій основними концепціями є поняття об’єктів і класів, які взаємодіють між собою за допомогою повідомлень, є

Відмітьте вірні варіанти:

	логічне програмування
	структурне програмування
x	об’єктно-орієнтовне програмування
	функціональне програмування
	імперативне програмування

Запитання №6

Парадигма програмування, яка описує процес обчислення у вигляді інструкцій, які змінюють стан програми, є

Відмітьте вірні варіанти:

	функціональне програмування
x	імперативне програмування
	структурне програмування
	об’єктно-орієнтовне програмування
	логічне програмування

Запитання №7

Вкажіть принципи щодо організації ППЗ.

Відмітьте вірні варіанти:

x	Змістова частина ППЗ повинна представлятися у вигляді окремих об’єктів, що дозволить структурувати дані, забезпечити каталогізацію і пошук об’єктів за їх властивостями
	ППЗ має базуватися на використанні обєктно-орієнтованій моделі
x	ППЗ повинен бути простим у використанні і доступним для оволодіння людиною, яка має лише загальні навички роботи з комп’ютером
x	Інтерфейс ППЗ має бути відкритим для взаємодії з іншими інформаційними системами
x	Узгодження змістової частини ППЗ з міжнародними, державними і галузевими (освітянськими) стандартами

Запитання №8

Вкажіть способи визначення вимог до ППЗ.

Відмітьте вірні варіанти:

	Вимоги керовані розробником
x	Вимоги контрольовані користувачем
x	Вимоги незалежні від користувача
x	Вимоги керовані користувачем

Запитання №9

Встановіть порядок етапів проектування та розробки ППЗ.

Проставте номери в правильній послідовності:

5	Програмування ППЗ
3	Добір змісту навчального матеріалу для ППЗ
1	Визначення цілей майбутнього ППЗ
4	Виокремлення тих дидактичних одиниць (законів, понять, факторів), які необхідні для створення та функціонування ППЗ
2	Визначення вимог до майбутнього ППЗ
6	Тестування

Запитання №10

Дидактична вимога

Відмітьте вірні варіанти:

	передбачає аргументотвність педагогічної доцільності використання ППЗ
х	передбачає забезпечення науковості змісту ППЗ передбачає подання засобами програми науково достовірних відомостей
	передбачає специфіку відповідної науки, її понятійного апарату, особливості методів дослідження її закономірностей
	передбачає реалізаці. сучасних методів обробки інформації
	передбачає необхідність враховувати своєрідність і особливості конкретного навчального предмета

Запитання №11

Обґрунтування вибору тематики

Відмітьте вірні варіанти:

х	передбачає аргументотвність педагогічної доцільності використання
	передбачає реалізаці. сучасних методів обробки інформації
	передбачає необхідність враховувати своєрідність і особливості конкретного навчального предмета
	передбачає забезпечення науковості змісту ППЗ передбачає подання засобами програми науково достовірних відомостей
	передбачає специфіку відповідної науки, її понятійного апарату, особливості методів дослідження її закономірностей

Запитання №12

До вимог, які ставляться до ППЗ відносяться:

Відмітьте вірні варіанти:

1	х	естетичні вимоги
2	х	фізіологічно-гігієнічні вимоги
3	х	технічні вимоги (програмно-технічні вимоги)
4	х	вимоги до оформлення документації
5	х	ергономічні вимоги

Запитання №13

Встановіть відповідність між терміном показниа якості ППЗ та відповідним означенням.

1) Функціональність ППЗ
2) Надійність ППЗ
3) Зручність використання (практичність)
4) Ефективність (продуктивність)
5) Мобільність ППЗ

Проставте номери в правильній послідовності:

1	1	надання користувачу можливості, за рахунок ППЗ, виконувати набір функцій, які задовольняють завданням чи можливим потребам користувачів у відповідній предметній галузі
2	3	характеристика ППЗ, згідно якої мінімізуються зусилля користувача щодо підготовки вхідних даних та аналізу отриманих результатів, а також викликаються позитивні емоції при його використанні
3	2	надання користувачу можливості, за рахунок ППЗ, безвідмовно виконувати визначені функції при заданих умовах протягом заданого періоду часу
4	5	збереження працездатності ППЗ при його переміщенні з одного апаратного і програмного оточення в інше без особливих зусиль
5	4	відношення рівня послуг, які отримує користувач при використанні ППЗ до об’єму використаних ресурсів комп’ютера

Запитання №14

Методична вимога

Відмітьте вірні варіанти:

1		передбачає аргументотвність педагогічної доцільності використання ППЗ
2	x	передбачає специфіку відповідної науки, її понятійного апарату, особливості методів дослідження її закономірностей
3	x	передбачає реалізацію сучасних методів обробки інформації
4		передбачає забезпечення науковості змісту ППЗ передбачає подання засобами програми науково достовірних відомостей
5	x	передбачає необхідність враховувати своєрідність і особливості конкретного навчального предмета