ТЕХНОЛОГІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ
ДО САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ ТА ВИКОНАННЯ СЕМЕСТРОВИХ ЗАВДАНЬ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Хіміко-технологічний факультет

ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ - 2. ТЕХНОЛОГІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ ДО САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ ТА ВИКОНАННЯ СЕМЕСТРОВИХ ЗАВДАНЬ

для студентів напряму підготовки
6.050202 – «Автоматизація та комп'ютерно-інтегровані технології»

Затверджено засіданням каф. Кібернетики ХТП НТУУ "КПІ", протокол № 12 від 13.06.2016 р.

Київ – 2016

ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ - 2. ТЕХНОЛОГІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ: МЕТОДИЧНІ ВКАЗІВКИ ДО САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ ТА ВИКОНАННЯ СЕМЕСТРОВИХ ЗАВДАНЬ для студентів напряму підготовки 6.050202 – «Автоматизація та комп'ютерно-інтегровані технології» [Електронний ресурс] / [уклад. Бендюг В. І., Комариста Б. М., Бондаренко О.С.]. – К: 2016. – 67 с. Систем. вимоги: Intel Core 2; 1 Gb RAM; Windows 7; Adobe Acrobat – Назва з екрану.

Затверджено засіданням каф. Кібернетики ХТП НТУУ "КПІ", протокол № 2 від 03.02.2016 р.

Електронне навчальне видання

ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ - 2. ТЕХНОЛОГІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ ДО САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ ТА ВИКОНАННЯ СЕМЕСТРОВИХ ЗАВДАНЬ

для студентів напряму підготовки
6.050202 – «Автоматизація та комп'ютерно-інтегровані технології»

Укладачі: Бендюг Владислав Іванович, канд. техн. наук, доцент
Комариста Богдана Миколаївна, канд. техн. наук
Бондаренко Олена Сергіївна, канд. техн. наук, доцент

Відповідальний редактор О. О. Квітка, канд. техн. наук, доцент

За редакцією укладачів

Зміст

Вступ

Згідно робочого навчального плану кредитний модуль «*Прикладне програмне забезпечення - 2. Технології об'єктно-орієнтованого програмування*» дисципліни «*Прикладне програмне забезпечення*» викладається студентам четвертого року підготовки ОКР «бакалавр» напряму підготовки *6.050202 - Автоматизація та комп'ютерно-інтегровані технології* у сьомому навчальному семестрі. Матеріал кредитного модуля базується на дисциплінах «Комп'ютерні технології та програмування - 1. Основи алгоритмізації», «Комп'ютерні технології та програмування - 2. Програмування типових задач», «Комп'ютерна техніка та організація обчислювальних робіт», «Комп'ютерні технології та програмування - 3. Розробка інтерфейсу користувача», «Інформаційні системи та комплекси». Знання уміння та навички, одержані під час вивчення даного модуля, у подальшому використовуються в усіх дисциплінах, які потребують програмної реалізації розрахунків на комп'ютері, і в першу чергу – в курсах «Ідентифікація та моделювання об'єктів автоматизації», «Основи проектування систем автоматизації і систем керування експериментом», «Методи штучного інтелекту та їх застосування в хімічній технології», «Прикладне програмне забезпечення - 3. Проектування програмних доданків», в курсових і дипломних роботах і проектах.

Метою навчальної дисципліни є формування у студентів здібностей:

КСП-24 – вміння застосовувати комп'ютерну техніку для вирішення технічних задач;

КСП-25 – вміння використовувати комп'ютерно-інтегровані технологічні та інформаційні системи;

КСП-26 – вміння розробляти, тестувати та застосовувати програмне забезпечення для вирішення прикладних задач.

Згідно з вимогами освітньо-професійної програми студенти після засвоєння навчальної дисципліни мають продемонструвати такі результати навчання:

знання:

- стандартів сучасних об'єктно-орієнтованих мов програмування;
- інтерфейсу середовища програмування Microsoft Visual Studio;
- сучасних методи конструювання програм в тому числі засобами об'єктно-орієнтованої мови програмування C++;

- ефективних методів збереження і обробки інформації;

уміння:

- самостійно розробляти ефективні алгоритми і програми на сучасній алгоритмічній мові для вирішення поставленої задачі;
- налагоджувати програми на ПК до надійної працездатності в ОС Windows;

досвід:

- виконувати налаштування середовища програмування для ефективної роботи;
- оформляти розроблені програми згідно вимогам ЄСПД.

1. Структура кредитного модуля

Таблиця 1.1 – Структура кредитного модуля дисципліни

<table>
<tr><th>Галузь знань, напрям підготовки, освітньо-кваліфікаційний рівень</th><th>Загальні показники</th><th>Характеристика кредитного модуля</th></tr>
<tr><td rowspan="2">Галузь знань
<u>0502 – Автоматика і управління</u>
(шифр і назва)</td><td rowspan="2">Назва дисципліни, до якої належить кредитний модуль
<u>Прикладне програмне</u> забезпечення</td><td>Форма навчання
<u>денна</u>
(денна / заочна)</td></tr>
<tr><td rowspan="2">Статус кредитного модуля
<u>за вибором ВНЗ</u>
(нормативний
або за вибором ВНЗ/студентів)</td></tr>
<tr><td>Напрям підготовки
<u>6.050202 Автоматизація та комп’ютерно-інтегровані технології</u>
(шифр і назва)</td><td>Кількість кредитів ECTS
<u>4</u></td></tr>
<tr><td>Спеціальність
<u>усі</u>
(шифр і назва)</td><td>Кількість розділів <u>2</u></td><td>Цикл до якого належить кредитний модуль
<u>II.2. Дисципліни вільного вибору студентів</u></td></tr>
<tr><td rowspan="2">Спеціалізація
<u>-</u>
(назва)</td><td rowspan="2">Індивідуальне завдання:
<u>ДКР</u>
(вид)</td><td>Рік підготовки <u>4</u></td></tr>
<tr><td>Семестр сьомий</td></tr>
<tr><td rowspan="4">Освітньо-кваліфікаційний рівень
<u>бакалавр</u></td><td rowspan="2">Загальна кількість годин
<u>120</u></td><td>Лекції
<u>18</u> год.</td></tr>
<tr><td>Практичні (семінарські)
<u>-</u></td></tr>
<tr><td rowspan="2">Тижневих годин:
аудиторних – <u>3</u>
СРС – <u>2,7</u></td><td>Комп’ютерний практикум
<u>54</u> год.</td></tr>
<tr><td>Самостійна робота
<u>48</u> год.
в тому числі на виконання індивідуального завдання
<u>7</u> год.</td></tr>
<tr><td colspan="2">Вид та форма семестрового контролю
<u>залік</u>
(екзамен / залік / диф. залік;
усний / письмовий / тестування тощо)</td></tr>
</table>

2. Рейтингова система оцінювання результатів навчання

Рейтинг студента з дисципліни складається з балів, що він отримує за:

1) Виконання та захист комп'ютерних практикумів;
2) Написання модульної контрольної роботи;
3) Виконання домашньої контрольної роботи;

Шкала балів за відповідні рівні оцінювання з кожного виду контролю. З урахуванням межових значень 0,9 – 0,75 – 0,6 – 0 маємо такий розподіл:

а) Комп'ютерні практикуми:
«відмінно» –7 балів;
«добре» – 5-6 балів;
«задовільно» – 4 балів;
«незадовільно» – 0 балів.
б) ДКР:
«відмінно» – 18-20 балів;
«добре» – 15-17 балів;
«задовільно» – 13-14 балів;
«незадовільно» – 0 балів.
в) МКР:
«відмінно» – 15 – 17 балів;
«добре» – 13 – 14 балів;
«задовільно» – 11 - 12 балів;
«незадовільно» – 0 балів.

Контрольна перевірка: студент, який отримав мінімальні позитивні бали за всіма контролями, матиме у підсумку не менше 60 балів.

$$4 \times 9 + 13 + 11 = 60 \text{ балів.}$$

Система рейтингових (вагових) балів та критерії оцінювання

1. Комп'ютерні практикуми.

Ваговий бал – 7.

Рейтингові бали комп'ютерного практикуму складаються з балів за підготовку та виконання практикуму (від 1 до 2), балів за оформлення протоколу практикуму (від 1 до 2 балів) і балів за захист практикуму (від 2 до 3). Таким чином за результатами практикуму студент може отримати від 4 до 7 балів.

За ***виконання роботи*** бали виставляються наступним чином:

- ∑ робота повністю і вірно виконана у відведений час – 2 бали;
- ∑ робота виконана у відведений час, але не повністю висвітлює деякі аспекти завдання – 1 бал;
- ∑ робота виконана невчасно або має суттєві недоліки – 0 балів.

За ***оформлення протоколу практикуму*** бали виставляються наступним чином:

- ∑ протокол відповідає вимогам, оформлений охайно, без виправлень і помарок (допускається не більше 1 виправлення на 1 сторінці протоколу) – 2 бали;
- ∑ протокол відповідає вимогам, проте є певні недоліки – 1 бал;
- ∑ протокол не відповідає основним вимогам до оформлення – 0 балів.

За ***захист практикуму*** бали виставляються наступним чином:

- ∑ студент вірно і повністю відповів на всі поставлені йому запитання (виконав надані для захисту роботи завдання) – 3 бали;
- ∑ студент відповів на запитання в цілому вірно, проте при відповідях припускався несуттєвих помилок – 2 бали;
- ∑ студент припустився помилок при відповідях на більшість питань або взагалі не відповів на більшу частину запитань – (0 балів);
- ∑ при отриманні 0 балів за захист, студент має повторно захищати лабораторну роботу на наступному занятті.

2.Модульний контроль

Ваговий бал – 17.

Модульна контрольна робота являє собою практичне завдання – створення консольного додатку для вирішування поставленої задачі згідно отриманого варіанту протягом відведеного часу. Модульна контрольна робота проводиться після вичитування основної частини лекцій курсу. За результатами виконання роботи для позитивної оцінки студент може отримати від 11 до 17 балів. Оцінювання такої роботи проводиться за наступною шкалою:

- ∑ студент повністю вірно і у повному обсязі виконав завдання (15 - 17 балів);
- ∑ студент вірно і у повному обсязі виконав завдання проте припустився несуттєвих помилок (13 - 14 балів);
- ∑ студент виконав основну частину завдання проте припустився помилок та виконав завдання не в повному обсязі (11 - 12 балів);
- ∑ студент не виконав основної частини завдання, або припустився суттєвих помилок (0 балів);
- ∑ студент не з'явився без поважних причин на модульний контроль (– 2 бали).

В разі, якщо студент не закінчив виконання роботи вчасно, оцінюється та частина, яка фактично виконана.

3.Домашня контрольна робота

Ваговий бал – 20.

Домашня контрольна робота являє собою практичне завдання – створення програмного модулю на мові програмування С++ згідно отриманого варіанту

індивідуального завдання. Оцінювання такої роботі проводиться за наступною шкалою:

- Σ студент повністю вірно і у повному обсязі виконав поставлене завдання та відповів на додаткові питання (18-20 балів);
- Σ студент повністю вірно і у повному обсязі виконав поставлене завдання але не відповів на додаткові питання (від 15 до 17 балів);
- Σ студент в цілому вірно та повністю виконав поставлене завдання але припустився несуттєвих помилок (від 13 до 14 балів);
- Σ студент виконав завдання неповністю та (або) припустився при виконанні суттєвих помилок (0 балів);
- Σ студент не здав домашню контрольну роботу у відведений час без поважної причини (– 1 бал за кожен тиждень затримки).

Штрафні та заохочувальні бали за *:

- Вдосконалення дидактичного матеріалу до комп'ютерного практикуму +1 бал;
- Вдосконалення дидактичного матеріалу до домашньої контрольної роботи +5 балів;
- Недопущення до комп'ютерного практикуму у зв'язку з незадовільним вхідним контролем (відсутність протоколу, недостатня підготовка до практикуму) –1 бал;
- Відсутність на комп'ютерному практикуму без поважної причини –1 бал;
- Відсутність у студента конспекту під час перевірки на лекційному занятті –1 бал;

Умови допуску до заліку

Умовами допуску до заліку є:

- Захист 9-ти комп'ютерних практикумів на позитивну оцінку (4 і більше балів);
- Написання модульної контрольної роботи на позитивну оцінку (11 або більше балів);
- Виконання домашньої контрольної роботи на позитивну оцінку (13 або більше балів);
- Наявність конспекту лекцій.

Умови отримання семестрової атестації

Календарна атестація студентів (на 8 та 14 тижнях семестрів) з дисциплін проводиться викладачами за значенням поточного рейтингу студента на час

атестації. Якщо значення цього рейтингу не менше 50 % від максимально можливого на час атестації, студент вважається задовільно атестованим. В іншому випадку в атестаційній відомості виставляється «незадовільно».

1-ша атестація

$$r_C = (4*7)*0,5 = 14$$

Якщо $r_C \geq 14$ студент вважається задовільно атестованим

2-га атестація

$$r_C = (7*7)*0,5 = 25$$

Якщо $r_C \geq 25$ студент вважається задовільно атестованим

Розрахунок шкали рейтингу:

Сума вагових балів контрольних заходів протягом семестру складає:

$$R_C = 7*9 + 17 + 20 = 100 \text{ балів.}$$

Таким чином, рейтингова шкала з дисципліни складає **R = 100 балів**.

Студенти, які набрали протягом семестру рейтинг менше **60 балів** зобов'язані виконувати залікову контрольну роботу.

Студенти, які набрали протягом семестру необхідну кількість балів (**60 балів і більше**), мають можливості отримати залікову оцінку (залік) відповідно до набраного рейтингу (табл. 1);

4. Залікова контрольна робота

Студенти, які наприкінці семестру мають рейтинг менше 60 балів, а також ті, хто хоче підвищити оцінку в системі ECTS, виконують ***залікову контрольну роботу***. При цьому до балів за МКР ($r_{МКР}$) та ДКР ($r_{ДКР}$) додаються бали за контрольну роботу і ця рейтингова оцінка є остаточною.

Завдання контрольної роботи складається з *практичного завдання по створенню програмного комплексу на мові С++. Додаткове питання* з тем лекційних занять отримують студенти, які були відсутні на певній лекції. Незадовільна відповідь з додаткового питання знижує загальну оцінку на 2 бали.

Оцінка за залікову контрольну роботу складається з 3-х частин: вірність і оптимальність розрахункового алгоритму програми – 23 балів; обробка та введення вихідних даних і виведення результатів розрахунку – 20 балів; оформлення інтерфейсу користувача, його зручність та оптимальність – 20 балів.

Оцінювання такої роботи проводиться за наступною шкалою:

- програмний комплекс відповідає всім вимогам та виконує вірно всі розрахунки – 57-63 бали;
- програмний комплекс працює вірно, проте, не відповідає певним вимогам до зручності програмного інтерфейсу – 47-56 балів;

- програмний комплекс працює в цілому вірно, проте, не відповідає більшості вимог до створення програмного інтерфейсу, або обробки вхідних даних та виведення результатів розрахунку – 38-46 балів;
- програмний комплекс працює не вірно, інтерфейс не відповідає сучасним вимогам, обробка вхідних даних та результатів не на належному рівні – 0 балів;
- якщо в програмному комплексі присутні помилки, або інтерфейс повною мірою не відповідає вимогам, то бали знімаються наступним чином:

∑ програмний комплекс працює, але не вірно виконує розрахунки – знімається 5 балів;

∑ не запрограмоване обробник помилок або не всі стандартні помилки при роботі програми перехоплюються програмою чи не виводяться відповідні конкретні повідомлення по певним типам помилок – знімається 3 бали;

∑ відсутні підказки для користувача при роботі з програмою – знімається 3 бали;

∑ не передбачена можливість завантаження даних з файлу чи збереження результатів роботи у файл за допомогою відповідних засобів мови С++ – знімається 3 бали;

∑ не оптимально виконана компоновка програмного коду, не виділені блоки введення даних і виведення результатів окремими логічними блоками - знімається 2 бали;

∑ допущено дрібні помилки при створенні чи роботі програмного комплексу – знімається 1 бал за кожну помилку.

В разі, якщо студент не закінчив виконання роботи вчасно, оцінюється та частина, яка фактично виконана.

Загальна сума балів за залікову контрольну роботу розподіляється наступним чином:

- ***«відмінно»,*** завдання виконане повністю без помилок (не менше 90% потрібної інформації) – ***57-63 бали***;
- ***«добре»,*** завдання виконане в достатньому обсязі з дрібними неточностями (не менше 75% обсягу завдання або незначні неточності) – ***47-56 балів***;
- ***«задовільно»,*** виконана більша частина завдання, або наявні деякі помилки (не менше 60% виконаного завдання та деякі помилки) – ***38-46 балів***;
- ***«незадовільно»,*** незадовільне виконання завдання – ***0 балів***.

$$R = r_{МКР} + r_{ДКР} + r_{ЗКР}$$

Сума балів за виконання залікової контрольної роботи та МКР і ДКР переводиться до залікової оцінки згідно з таблицею 2.1.

Таблиця 2.1 – Системи переведення балів до залікової оцінки

Бали, R	***ECTS оцінка***	***Залікова оцінка***
95-100	*A*	***Зараховано***
85-94	*B*	
75-84	*C*	
65-74	*D*	
60-64	*E*	
Менше 60	*Fx*	***Незараховано***
МКР не зараховано	***F***	***Не допущено***

3. Підготовка до лекцій

Загальні рекомендації

Лекція, як вид навчального заняття, не передбачає окремих процедур контролю підготовленості студентів. Проте це не означає, що студент має прийти на лекцію непідготовленим. Готуючись до кожної лекції з дисципліни «Технології об'єктно-орієнтованого програмування» студент повинен:

- ознайомитися з темою лекції та переліком питань, які будуть розглядатися на лекції;
- повторити попередньо освоєний матеріал, який потрібен для успішного засвоєння лекції;
- підготувати запитання з теми лекції, відповіді на які студент хотів би отримати;
- самостійно освоїти питання, які винесені на самостійне опрацювання за темою лекції.

Теми лекцій і перелік питань, що на них будуть розглядатися, наведені у цьому розділі. Для кожного питання наведені посилання на літературні джерела. Також наведений перелік матеріалів, знання яких є передумовою продуктивної роботи студента на лекції.

Запитання до теми лекції студент може готувати як письмово, так і в усній формі. Ймовірно, що в процесі читання лекції викладач дасть відповіді на питання, що зацікавили студента і частину питань буде, таким чином, знято. Ті запитання, що не знайшли відповіді у самій лекції, а також ті, відповіді на які не зрозумілі студенту, настійливо рекомендується задати викладачеві у кінці лекції або на консультаціях.

Після прочитання кожної лекції студентам пропонується декілька питань на самостійне опрацювання за темою. Такі питання та перелік літературних джерел для їх освоєння наведені у цьому розділі.

Лекція 1. Створення простої програми на C++

Питання, які розглядаються на лекції:

- Створення консольного додатку Win32.
- Компіляція та запуск.
- Стандартні оператори введення-виведення.
- Введення даних з файлу.
- Виведення інформації у файл.
- Використання коментарів.

*Література: [*1, с. 25-36*].*

Питання, які виносяться на самостійне опрацювання:

Об’єкти, типи і значення:

- введення і тип;
- операції і оператори;
- присвоєння і ініціалізація;
- імена; типи і об’єкти;
- безпечні перетворення;
- небезпечні перетворення.

Література: [2, с. 93-120].

Лекція 2. Засоби керування ходом виконання програми

Питання, які розглядаються на лекції:

Засоби керування ходом виконання програми.

- Оператор while.
- Оператор for.
- Введення невідомої кількості даних.
- Оператор if.

Введення в класи

- Поняття класу.
- Перший погляд на функції-члени.

*Література: [*1, с. 37-51*].*

Питання, які виносяться на самостійне опрацювання:

Початкові відомості про C++:

- функція main();
- коментарі в мові C++;
- препроцесор C++ і файл iostream;
- форматування вихідного коду C++;
- стиль форматування вихідного коду програм на C++.

*Література: [*3, с. 44-50, 56-57*].*

Лекція 3. Змінні та базові типи

Питання, які розглядаються на лекції:

Прості вбудовані типи:

- арифметичні типи;
- перетворення типів;
- літерали.

Змінні:

- визначення змінних;
- об’явлення та визначення змінних;
- ідентифікатори;
- область видимості імен.

Складені типи:

- посилання;
- покажчики;
- поняття опису складених типів.

Специфікатор const:

- посилання на константу;
- покажчики та специфікатор const;
- специфікатор const верхнього рівня;
- змінні constexpr і константні вирази.

Робота з типами:

- псевдоніми типів;
- специфікатор типу auto;
- специфікатор типу decltype.

Визначення власних структур даних:

- визначення типу;
- використання визначеного типу;
- створення власних файлів заголовку.

*Література: [*1, с. 61-118*].*

Питання, які виносяться на самостійне опрацювання:

Початкові відомості про C++:

- імена заголовочних файлів;
- простори імен.

Тип даних char:

- символи і малі цілі числа.

Числа з плаваючою комою:

- запис чисел з плаваючою комою;
- типи даних з плаваючою комою;
- константи з плаваючою комою;
- переваги та недоліки типів з плаваючою комою.

*Література: [*3, с. 51-52, 93-101, 103-109*]*

Лекція 4. Типи string, vector та масиви

Питання, які розглядаються на лекції:

Побудова імен і визначення using.

Бібліотечний тип string:

- визначення та ініціалізація рядків;
- операції з рядками;
- робота з символами рядка.

Бібліотечний тип vector:

- визначення та інінціалізація векторів;
- додавання елементів у вектор;
- інші оператори з векторами.

Знайомство з ітераторами:

- використання ітераторів;
- арифметичні дії з ітераторами.

Масиви:

- визначення та ініціалізація вбудованих масивів;
- доступ до елементів масиву;
- покажчики та масиви;
- символьні рядки у стилі С;
- взаємодія із застарілим кодом;
- багатомірні масиви.

*Література: [*1, с. 123-182*].*

Питання, які виносяться на самостійне опрацювання:

Вектор:

- збільшення вектору;
- числовий приклад;
- текстовий приклад.

Переліки:

- встановлення значень переліків;
- діапазони значень для переліків.

Покажчики і вільна пам’ять:

- об’явлення і ініціалізація покажчиків;
- покажчики і числа;
- виділення пам’яті за допомогою оператора new;
- вивільнення пам’яті за допомогою оператору delete;
- використання оператора new для створення динамічних масивів.

Покажчики, масиви та арифметика покажчиків:

- покажчики та рядки;
- використання оператора new для створення динамічних структур;
- автоматичне, статичне та динамічне виділення пам’яті.

*Література: [*2, с. 148-152; 3, с. 148-180*]*

Лекція 5. Вирази

Питання, які розглядаються на лекції:

Основи роботи з виразами:

- фундаментальні концепції;
- пріоритет і порядок;
- порядок обчислення.

Арифметичні оператори.

Логічні оператори та оператори відношення.

Оператори присвоєння.

Оператори інкременту та декременту.

Оператори доступу до членів.

Умовний оператор.

Побітові оператори.

Оператор sizeof.

Оператор кома.

Перетворення типів:

- арифметичні перетворення;
- інші неявні перетворення;
- явні перетворення.

Таблиця пріоритетів операторів.

*Література: [*1, с. 187-228*].*

Питання, які виносяться на самостійне опрацювання:

Арифметичні операції в мові С++:

- пріоритет операцій і асоціативність;
- різновиди операцій ділення;
- операції ділення за модулем;
- перетворення типів даних;
- перетворення даних у виразах.

*Література: [*3, с. 110-120*]*

Лекція 6. Оператори

Питання, які розглядаються на лекції:

Прості оператори.

Операторна область видимості.

Умовні оператори:

- оператор if;
- оператор switch.

Ітераційні оператори:

- оператор while;
- традиційний оператор for;
- серійний оператор for;
- оператор do while.

Оператори переходу:

- оператор break;
- оператор continue;
- оператор goto.

Блок try та обробка виключень:

- оператор throw;
- блок try;
- стандартні виключення.

*Література: [*1, с. 233-263*].*

Питання, які виносяться на самостійне опрацювання:

Помилки:

- джерела помилок;
- помилки під час компіляції;
- помилки під час редагування зв’язків;
- помилки під час виконання програми;
- виключення;

- логічні помилки;
- оцінка;
- відладка;
- тестування.

*Література: [*2, с. 161-200*]*

Лекція 7. Функції

Питання, які розглядаються на лекції:

Основи функцій:

- локальні об’єкти;
- об’явлення функцій;
- роздільна компіляція.

Передача аргументів:

- передача аргументу по значенню;
- передача аргументу за посиланням;
- константні параметри та аргументи;
- параметри у вигляді масиву;
- функції зі змінною кількістю параметрів.

Типи значень, які повертає функція та оператор return:

- функції, які не повертають значення;
- функції, які повертають значення;
- повернення покажчика на масив.

Перевантажувані функції.

Спеціальні засоби:

- аргументи за замовчуванням;
- вбудовані функції та функції constexpr.

Перетворення типів аргументів.

Покажчики на функції.

*Література: [*1, с. 267-325*].*

Питання, які виносяться на самостійне опрацювання:

Функції мови С++:

- функції і рядки у стилі С;
- функції і структури;
- рекурсія.

Шаблони функцій:

- перевантажувані шаблони;
- явні спеціалізації;
- спеціалізації третього покоління;
- створення екземплярів і спеціалізація;
- вибір функцій;
- використання правил часткового впорядкування.

Класи і функції:

- проектування і об’явлення.

*Література: [*3, с. 305-319, 358-374; 4, с. 79-118*].*

Лекція 8. Класи

Питання, які розглядаються на лекції:

Визначення абстрактних типів даних:

- розробка власного класу;
- визначення класу;
- визначення функцій, які не є членами класу але поєднані з ним;
- конструктори;
- копіювання, присвоєння та видалення.

Керування доступом та інкапсуляція.

Додаткові засоби класу:

- члени класу;
- функції, що повертають покажчик *this;
- типи класів;
- дружні відношення.

*Література: [*1, с. 331-391*].*

Питання, які виносяться на самостійне опрацювання:

Робота з класами:

- перевантаження операцій;
- використання дружніх структур;
- клас vector;
- автоматичне перетворення і приведення типів для класів.

*Література: [*3, с. 482-535; 4, с. 119-144*].*

Лекція 9. Класи

Питання, які розглядаються на лекції:

Область видимості класу.

Застосування конструкторів:

- список ініціалізації конструктору;
- делегуючий конструктор;
- роль стандартного конструктора;
- неявне перетворення типів класу;
- агрегатні класи;
- літеральні класи;
- статичні члени класу.

*Література: [*1, с. 331-391*].*

Питання, які виносяться на самостійне опрацювання:

Класи і функції:

- реалізація.

*Література: [*3, с. 482-535; 4, с. 119-144*].*

4. Підготовка до комп'ютерних практикумів

Загальні вимоги та рекомендації

Комп'ютерні практикуми проводяться у комп'ютерних класах з окремими групами студентів. У випадку великої кількості студентів у групі, вона може поділятися на дві підгрупи і заняття проводяться у двох класах одночасно. Метою комп'ютерних практикумів є вироблення та закріплення умінь та навичок програмування мовою С++.

Студент зобов'язаний ретельно готуватися до кожного з комп'ютерних практикумів. Самостійна робота студента при підготовці до комп'ютерного практикуму з дисципліни «Технології об'єктно-орієнтованого програмування» передбачає:

- освоєння теоретичного матеріалу з теми практикуму на основі лекцій та літературних джерел;
- відповіді на питання контролю підготовленості до практикуму;
- підготовку звіту з практикуму.

Перелік теоретичного матеріалу, який необхідно освоїти та питання для контролю підготовленості до кожного з практикумів наведені у цьому розділі. Звіт з комп'ютерного практикуму оформлюється на аркушах формату А4 і має включати в себе:

- *титульний аркуш* з зазначенням номеру практикуму, його теми, групи і прізвища студента, який виконав роботу та прізвища викладача, який її перевірив (зразок титульного аркуша наведено в додатку А);
- *мету* та *завдання* практикуму;
- *короткі теоретичні відомості* для висвітлення питань, яким присвячена робота;
- *порядок виконання* завдань практикуму, з зазначенням виконуваних дій та операцій;
- *результати виконання* завдань у роздрукованому вигляді;
- *код програмного модуля*.

Перші три пункти цього переліку готуються студентом заздалегідь і пред'являються викладачу на початку практикуму. Це є необхідною умовою його успішного виконання. Результати практикуму друкуються, очевидно, після виконання завдань і перевірки їх викладачем. Результати роботи створеної

програми, а також код програмного модулю слід друкувати як частину протоколу на аркушах формату А4.

Виконання кожного з комп'ютерних практикумів проводиться за методикою, викладеною у відповідних методичних вказівках [5]. Порядок оцінювання комп'ютерних практикумів наведено в розділі 2.

Комп'ютерний практикум 1. Створення консольного додатку

Мета: ознайомитись з навичками роботи в командному режимі та принципами створення *консольного додатку* C++. Засвоїти методи *компіляції*, *запуску* та *відлагоджування* програми. Вивчити оператори *потоків введення-виведення* в консольному додатку.

Завдання: створити консольний додаток для відпрацювання основних навичок створення та компіляції додатку, а також реалізувати поставлену задачу згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Що таке функція `main()`, з чого вона складається?
2) Який тип даних має повертати функція `main()`?
3) Що таке вбудований тип даних?
4) Що таке тіло функції, з чого воно складається?
5) Що таке оператор `return`, для чого він використовується?
6) Що таке потік введення чи виведення?
7) Що таке стандартне введення та стандартне виведення?
8) Що таке об'єкт `cerr` та `clog` і для чого вони використовуються?
9) Що таке директива `#include` та для чого вона використовується?
10) Що таке оператор виведення та для чого він використовується?
11) Що таке оператор введення та для чого він використовується?
12) Що таке маніпулятор та для чого він використовується?
13) Що таке простір імен?
14) Що таке оператор області видимості та для чого він використовується?
15) Що таке ініціалізація?

Комп'ютерний практикум 2. Оператор if та логічні оператори

Мета: ознайомитись з використанням *математичних функцій* в C++, вивчити роботу *умовного оператора if*, розглянути можливості застосування *логічних операторів* та *операторів відношення*.

Завдання: створити консольний додаток для реалізації обчислень поставленої задачі згідно отриманого варіанту завдання з використанням умовного оператора *if* та математичних функцій.

Питання для контролю підготовленості до практикуму:

1) Опишіть структуру та принцип роботи оператора `if`?
2) Унарні та прані оператори. В чому їх відмінність?
3) Перерахуйте арифметичні оператори згідно з їх пріоритетом.
4) Перерахуйте логічні оператори та оператори відношення.
5) Перерахуйте складені оператори присвоєння та поясніть принцип їх дії.
6) Що таке оператор прирощення та оператор зменшення? Поясніть їх використання на прикладі.
7) Що таке префіксний оператор та постфіксний оператор? Поясніть відмінність між ними.
8) Що таке об'явлення `using` та для чого воно використовується?
9) Що потрібно для використання в програмі математичних функцій? Перерахуйте основні математичні функції.

Комп'ютерний практикум 3. Засоби керування процесом виконання програми

Мета: ознайомитись з операторами для зміни послідовності виконання програми, вивчити роботу *оператора ітераційного циклу while*, вивчити роботу *оператора арифметичного циклу for*, вивчити роботу *генератора випадкових чисел*.

Завдання: створити консольний додаток для реалізації обчислень поставленої задачі згідно двох частин отриманого варіанту завдання з використанням операторів циклу *while* та *for*, а також умовного оператора *if*.

Питання для контролю підготовленості до практикуму:

1) Опишіть структуру та принцип роботи оператора `while`?
2) Що таке блок? Наведіть приклад.
3) Опишіть структуру та принцип роботи оператора `for`?
4) З чого складається заголовок оператора `for`?
5) В чому відмінність між операторами циклу `for` та `while`?
6) Для чого використовуються об'єкти класу `default_random_engine`?
7) Як згенерувати випадкові цілі числа в заданому діапазоні?
8) Як згенерувати випадкові дійсні числа в заданому діапазоні?
9) Як забезпечити генерування різних послідовностей випадкових чисел при кожному запуску програми?

Комп'ютерний практикум 4. Змінні та базові типи

Мета: ознайомитись з символьними та чисельними обчисленнями засобами C++; вивчити існуючі типи даних, структуровані типи даних (*вказівники, посилання, створення власної структури даних*); навчитись створювати *файли заголовку*; відпрацювати засоби візуалізації розрахунків в C++. Навчитись використовувати кирилицю в консолі: коректне розпізнавання введених символів кирилицею; коректне виведення в консолі символів кирилицею.

Завдання: створити консольний додаток для реалізації обчислень поставленої задачі згідно двох частин отриманого варіанту завдання з використанням створеної структури даних та покажчиків і посилань.

Питання для контролю підготовленості до практикуму:

1) Перерахуйте арифметичні типи мови C++ та вкажіть, які значення вони можуть зберігати.
2) Що таке знакові та беззнакові цілочиселні типи? Перерахуйте беззнакові цілочисельні типи.
3) Розкажіть що таке автоматичне перетворення типів та за якими правилами воно виконується.
4) Що таке змінна? З чого складається визначення змінної?
5) Що таке ініціалізація та для чого вона потрібна?
6) Що таке ідентифікатори? Які існують правило до створення ідентифікатора?
7) Що таке область видимості? Яка область видимості у об'єктів класу?
8) Що таке складений тип? Які складені типи ви знаєте?
9) Що таке посилання? Які правила використання та обмеження існують для посилань?
10) Що таке покажчик? Які правила використання існують для покажчиків? Як звернутись до значення покажчика?
11) Що таке специфікатор типу `auto`? Коли і як його слід використовувати?
12) Що таке специфікатор типу `decltype`? Коли і як його слід використовувати?
13) Що таке власна структура даних або клас? З чого складається визначення класу?
14) Що таке змінні-члени? Що таке внутрішньокласовий ініціалізатор?
15) Для чого потрібні файли заголовку? Як підключити заголовок до програми? Як організувати захист заголовку?

16) Що таке змінні препроцесора? Які правила створення імен змінних препроцесору та файлів заголовку загальноприйняті? Як працює директива `#include`?
17) Які таблиці кодування використовуються у `Windows`? В чому їх відмінності?
18) Що потрібно для коректного відображення кирилиці у консольному додатку? Для чого використовують файл заголовку `windows.h`? Для чого потрібні функції `SetConsoleCP()` і `SetConsoleOutputCP()`?

Комп'ютерний практикум 5. Типи string та масиви

Мета: ознайомитись з бібліотечним типом *string*: операції з рядками; робота з символами рядка. Засвоїти використання *оператору індексування* для роботи з *структурами даних*. Ознайомитись з використанням *ітераторів* для типу *string*. Вивчити роботу з *масивами*: *визначення* та *ініціалізація*; доступ до елементів; застосування *покажчиків*; *динамічні масиви*.

Завдання: створити консольний додаток для роботи з рядковим текстом, а також проведення розрахунків за допомогою одномірного масиву для вирішення поставленої задачі згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Для чого потрібний тип `string`, як створити об'єкт даного типу у програмі?
2) Як можна ініціалізувати об'єкти типу `string`? Наведіть приклади способів ініціалізації.
3) Що таке пряма ініціалізація та ініціалізація копією?
4) Перерахуйте основні операції визначені для класу `string`.
5) Як можна організувати циклічне введення невизначеної кількості даних?
6) Для чого використовується функція `getline()`, в чому її відмінність від оператора >>?
7) Для чого потрібен допоміжний тип даних `size_type` у класі `string`? Як створити об'єкт даного типу?
8) Перерахуйте функції для виконання операцій з символами рядка та їх призначення.
9) Що таке оператор індексування? Для чого він потрібен?
10) Що таке ітератори? Для чого вони використовуються?
11) Для чого потрібні функції-члени `begin()` та `end()` у контейнерах?
12) Який тип мають ітератори? Як визначити об'єкт даного типу?

13) Перерахуйте та поясніть стандартні операції з ітераторами.
14) Як за допомогою ітератора звернутись до значення контейнеру? Чому в циклі `for` при перевірці умови досягнення останнього елементу контейнера використовують оператор `!=` замість оператора `<`?
15) Як організувати переміщення по елементам контейнеру за допомогою ітераторів?
16) Що таке масив? Як об'явити масив?
17) Що таке багатомірний масив? Як можна ініціалізувати масив? Як отримати доступ до елементів масиву за допомогою оператора індексації?
18) Що таке динамічний масив? Як об'явити динамічний масив? Як організувати доступ до елементів масиву за допомогою покажчиків?
19) Для чого потрібен оператор `delete`? Яка форма його запису?

Комп'ютерний практикум 6. Тип vector

Мета: вивчити бібліотечний тип *vector*: *визначення* і *ініціалізація*; додавання елементів; інші операції. Засвоїти використання *оператору індексування* для типу *vector*. Ознайомитись з використання *ітераторів* для типу *vector*. Опрацювати використання двомірних *векторів*.

Завдання: створити консольний додаток для організації роботи з одномірним та двомірним масивом з використанням бібліотечного типу *vector* для реалізації поставленої задачі згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Що таке вектор? Що потрібно для використання векторів у програмі?
2) Що таке шаблон класу? Що таке створення екземпляру шаблону? Наведіть приклади.
3) Наведіть способи ініціалізації об'єктів класу `vector`.
4) Що таке ініціалізація переліком та ініціалізація значення для об'єкту типу `vector`? В чому відмінність використання круглих та фігурних дужок під час ініціалізації об'єкту типу `vector`, наведіть приклади?
5) Перерахуйте основні операції з векторами.
6) Поясніть як можна отримати доступ до елементів вектора за допомогою індексування та ітераторів? Наведіть приклади.
7) Що таке псевдонім типу та які методи створення псевдонімів ви знаєте? Наведіть приклади об'явлення псевдонімів.
8) Що таке оператор звернення до значення та оператор звернення до члену? Наведіть приклади.

9) Для чого потрібен оператор стрілки? Наведіть приклади.
10) Перерахуйте основні відмінності між масивами та векторами. Який з цих типів більш зручний у використанні?

Комп'ютерний практикум 7. Оператори

Мета: вивчити *умовні оператори*: оператор *if*, оператор *switch*; *ітераційні оператори*: цикл *do while*, *серійний оператор циклу for*; *оператори переходу*: *break*, *continue*. Засвоїти принципи застосування *обробки виключень*: робота з оператором *throw* та блоком *try*. Відпрацювати роботу зі *структурами даних:* використання оператора *struct*; створення та підключення файлів заголовку.

Завдання: створити консольний додаток для реалізації обчислень поставленої задачі згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Що таке оператор? Що таке оператор керування потоком виконання? Що таке оператор виразу?
2) Що таке пустий оператор? Які особливості його використання?
3) Що таке складений оператор? Що таке область видимості блоку, область видимості керуючої структури?
4) Що таке умовний оператор (`?:`), наведіть приклад його використання.
5) Що таке умовний оператор `switch`? Що таке блок `case` та блок `default`, для чого вони використовуються?
6) Що таке оператор `break`? Які особливості використання міток `case`? Наведіть приклад використання оператора `switch`?
7) Що таке ітераційні оператори? Що таке цикли з передумовою та післяумовою?
8) Що таке серійний оператор `for`? Опишіть принцип його роботи. В чому відмінність серійного оператора `for` від традиційного?
9) Що таке оператор `do while`? В чому його відмінність від оператора `while`? Наведіть приклад організації програми з можливістю повторної організації розрахунків по запиту користувача.
10) Опишіть особливості використання операторів `break` та `continue`, в чому їх відмінність?
11) Що таке виключення? Що таке обробка виключень?
12) Що таке оператор `throw`? Що таке блок `try`? Що таке розділ `catch`?
13) Які бібліотечні класи виключень ви знаєте, в яких заголовках вони визначені? Перерахуйте стандартні класи виключень заголовку `stdexcept`?

14) Для чого потрібна функція `what()`? Наведіть приклад її використання.
15) Як організувати власну структуру даних? Як створити файл заголовку для об'явлення класу?

Комп'ютерний практикум 8. Функції

Мета: вивчити принципи створення *функцій користувача* та роботи з ними: *визначення функції*; *тіло функції*; тип значення, яке повертає функція з використанням оператору *return*; *об'явлення функцій* у *файлі заголовку*; виклик функції; передача аргументів; створення програм за допомогою функцій.

Завдання: створити консольний додаток для обчислення значень функціональної залежності згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Що таке функція?
2) Що таке визначення функції, тип повернутого значення, параметри, тіло функції?
3) Що таке оператор виклику функції, з чого він складається?
4) Для чого потрібен оператор `return`?
5) Що таке аргументи функції? Для чого використовуються аргументи?
6) Що таке перелік параметрів функції? Як об'являються параметри функції? Чи може бути функція без параметрів? Чи може функція не повертати значення? Наведіть приклади.
7) Що таке область видимості імен та тривалість існування об'єкту?
8) Що таке локальні змінні? Чим вони відрізняються від глобальних?
9) Що таке автоматичний об'єкт? Що таке локальний статичний об'єкт? Наведіть приклади?
10) Що таке об'явлення функції? Що таке інтерфейс функції? Наведіть приклад.
11) Як створити зовнішню функцію? Як підключити зовнішню функцію користувача для використання у файлі вихідного коду?
12) Що таке роздільна компіляція та для чого вона потрібна?
13) Чим відрізняється передача аргументу за посиланням від передачі аргументу за значенням?
14) Скільки значень може повертати функція? Чи може функція повернути кілька значень, якщо так, то як це реалізувати? Наведіть приклади.
15) Як потрібно визначати параметр функції, який не повинен змінюватись? Наведіть приклади.

16) Що таке перевантажені функції? Які особливості їх використання? Наведіть приклади.

Комп'ютерний практикум 9. Класи

Мета: вивчити принципи створення власних *класів* та роботи з ними: *визначення класу*; створення *конструктору класу*; створення *інтерфейсу класу*; створення *реалізації класу*; *об'явлення функцій-членів класу*; виклик *функції-членів* класу; створення змінної типу власного класу; передача аргументів *функціям-членам* класу; створення програм з обчислювальною частиною, яка реалізована у власному класі та його *функціях-членах*.

Завдання: створити консольний додаток для розрахунку інтегралу методом Сімпсона згідно отриманого варіанту завдання.

Питання для контролю підготовленості до практикуму:

1) Що таке абстракція даних? Що таке абстрактний тип даних?
2) Що таке інтерфейс та реалізація класу?
3) Що таке інкапсуляція? Що таке специфікатори доступу?
4) В чому різниця між використанням ключових слів `struct` та `class`?
5) Що таке функції-члени? Де та як вони об'являються та визначаються?
6) Що таке область видимості класу? Наведіть приклад визначення функції-члену поза тілом класу.
7) Для чого використовується параметр `this`? Як звернутися до члену об'єкту класу? Наведіть приклади.
8) Як підключити допоміжні функції для використання класом без їх включення у клас?
9) Що таке конструктор, для чого він використовується?
10) Що таке стандартний конструктор? Що таке синтезований стандартний конструктор? Як об'явити стандартний конструктор?
11) Що таке перелік ініціалізації конструкутору? Для чого він потрібен?
12) Де потрібно визначати конструктор? З чого складається конструктор? Наведіть приклади об'явлення та визначення конструкторів?
13) Що таке дружні відносини? Як об'явити дружню функцію? Як зробити дружню функцію доступною для користувачів класу?

5. Самостійна робота

Перелік тем для самостійного вивчення

Потоки введення і виведення:

- модель потоку введення і виведення;
- файли;
- відкриття файлу;
- читання і запис файлу;
- обробка помилок введення-виведення;
- зчитування окремого значення;
- оператори введення, які визначені користувачем;
- оператори виведення, які визначені користувачем;
- стандартний тип введення;
- читання структурованого файлу.

*Література: [*2, с. 361-398*].*

Виведення на екран, використання графічних примітивів:

- використання бібліотеки графічного інтерфейсу користувача;
- координати;
- клас shape;
- використання графічних примітивів.

Графічні класи.

Проектування графічних класів.

*Література: [*2, с. 431-530*].*

Графічні функції і дані:

- побудова простих графіків;
- клас function;
- вісі;
- апроксимація;
- графічні дані.

Графічний інтерфейс користувача:

- альтернативи інтерфейсу користувача;

- просте вікно;
- додавання меню;
- відлагоджування інтерфейсу користувача.

*Література: [*2, с. 531-590*].*

6. Домашня контрольна робота

Вимоги до виконання та оформлення роботи

Домашня контрольна робота виконується студентом самостійно в час, вільний від занять. З усіх питань, які виникають у процесі виконання домашньої контрольної роботи, студент може звернутися до викладача у час, відведений для проведення консультацій.

Завдання на домашню контрольну роботу з дисципліни «Технології об'єктно-орієнтованого програмування» видається студентам після прослуховування переважної частини лекційного курсу, зазвичай на 8-9 тижні семестру. Видача завдань та закріплення варіантів домашньої контрольної роботи обов’язково фіксується у листі реєстрації індивідуальних завдань.

Час виконання роботи складає один місяць, рахуючи з дати видачі завдання. Протягом всього цього терміну студенти можуть звертатися до викладача за консультаціями щодо неї. Після завершення терміну виконання, консультації з питань домашньої контрольної роботи завершуються і студентам надається додатково 3-4 дні для здачі роботи. У разі, якщо студент не дотримав термінів здачі роботи, він отримує до свого семестрового рейтингу штрафні рейтингові бали (див. розділ 2).

Виконана робота здається викладачу в роздрукованому вигляді з додаванням електронної версії програмного модуля свого варіанту завдання. Роздрукований примірник домашньої контрольної роботи має складатись з:

1) *Титульного аркушу*;
2) *Завдання* (загальної частини та індивідуального завдання);
3) *Теоретичних відомостей* (коротко описуються застосовані при виконанні теоретичні аспекти);
4) *Ходу виконання* (описується створення програмного модулю, описується алгоритм роботи з програмним модулем, наводиться приклад застосування програмного модулю та результати розрахунку);
5) *Коду програмного модулю* (наводиться повний код програмного модуля з усіма окремими файлами модулів *.cpp та файлами заголовку *.h).

До оголошення оцінок за контрольну роботу студент повинен зберігати електронні варіанти проекту з вихідними файлами виконаного завдання.

Методичні рекомендації до виконання роботи

З метою закріплення отриманих знань та навичок, в рамках кредитного модулю передбачене виконання домашньої контрольної роботи. Домашня контрольна робота виконується за темою: «Створення та використання класів користувача». Для виконання роботи студентам видаються додому індивідуальні завдання. За результатами домашньої контрольної роботи робиться висновок про достатнє (недостатнє) володіння студентом матеріалом дисципліни.

Мета: вивчити принципи створення власних *класів* та роботи з ними: *визначення класу*; створення *конструктору класу*; створення *інтерфейсу класу*; створення *реалізації класу*; *об'явлення функцій-членів класу*; виклик *функції-члену* класу; створення змінної типу власного класу; передача аргументів *функціям-членам* класу; створення програм з обчислювальною частиною, яка реалізована у власному класі та його *функціях-членах*.

Завдання: створити консольний додаток який реалізує розрахунок з використанням математичного методу згідно отриманого варіанту завдання у вигляді класу.

Загальні вимоги.

1) Створити консольний додаток з ім'ям виду *PrizvischeDKPR*.
2) Програмний код модуля моє бути чітко структурований.
3) Імена об'єктів мають нести сенсові навантаження.
4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

1) Створити за допомогою *файлу заголовку* власний *клас даних*, який реалізуватиме розрахунок відповідним математичним методом (згідно варіанту).
2) Введення вхідних даних реалізувати в основному тілі програми.
3) Всі етапи обчислення реалізувати у власному класі.
4) В класі передбачити три конструктори: конструктор за замовчуванням; конструктор для ініціалізації класу всіма необхідними даними (коефіцієнти рівняння, межі інтегрування, точність інтегрування); для ініціалізації класу лише мінімально необхідними даними (наприклад, в реалізації класу передбачити ініціалізацію коефіцієнтів рівняння, меж інтегрування та точності інтегрування значеннями за замовчуванням);

5) В класі створити інтерфейс класу (відкриту для користувача частину) та реалізацію класу (інкапсульовану всередині класу його частину). Інтерфейсна частина класу та частина його реалізації повинні мати не менше однієї функції-члену кожна.
6) В програмі передбачити можливість вибору користувача: вводити всі вхідні дані (наприклад, коефіцієнти рівняння, межі інтегрування, точність інтегрування) для ініціалізації об'єкту класу за допомогою розширеного конструктора; вводити лише мінімально необхідні дані (наприклад, інтервал інтегрування) для виклику конструктора класу, який передає ініціалізацію інших вхідних даних значеннями за замовчуванням у блоці реалізації класу.
7) Передбачити виведення результатів розрахунку на кожній ітерації, а також знайденого результату з заданою точністю на екран ту у зовнішній файл.

Теоретичні відомості

Класи в мові C++ використовуються для визначення власних типів даних. Фундаментальними поняттями концепції класів є абстракція даних та інкапсуляція.

Абстракція даних (data abstraction) – програмний підхід, що заснований на розділенні інтерфейсу та реалізації. *Інтерфейс* (interface) класу складається з операцій, які користувач класу може виконати з його об'єктом. *Реалізація* (implementation) включає змінні-члени класу, тіла функцій, що складають інтерфейс, а також інші функції, котрі потрібні для визначення класу, проте не призначені для загального використання.

Інкапсуляція (encapsulation) забезпечує розділення інтерфейсу та реалізації класу. Інкапсульований клас приховує свою реалізацію від користувачів, які можуть використовувати інтерфейс, проте не мають доступу до реалізації класу.

Під *користувачами* класу в C++ маються на увазі розробники програмного коду, які використовують вже створений раніше іншими розробниками клас.

Клас, який використовує абстракцію даних та інкапсуляцію, називають *абстрактним типом даних* (abstract data type). Програмісти, які працюють з класом, не мають знати як внутрішньо працює цей тип. Вони можуть розглядати його як абстракцію.

Для забезпечення інкапсуляції в C++ використовують *специфікатори доступу* (access specifier).

- Члени класу, які визначені після *специфікатора* `public`, доступні для всіх частин програми. *Відкриті члени* (public member) визначають *інтерфейс класу*.

- Члени, які визначені після *специфікатору* `private`, є *закритими членами* (`private member`), вони доступні для функцій-членів класу, але не доступні для коду, який цей клас використовує. Розділи `private` інкапсулюють (приховують) реалізацію.

На найпростішому рівні *структура даних* (`data structure`) – це спосіб групування взаємопов'язаних даних та стратегії їх використання.

Визначення класу починається із *ключового слова* `struct`, яке супроводжується ім'ям класу та (можливо пустим) тілом класу. *Тіло класу* обмежується фігурними дужками і формує нову *область видимості*. Визначені у класі імена мають бути унікальними в межах класу, але поза класом вони можуть повторюватись.

```
struct MyStruct //Створення власної структури даних (класу)
{
	std::string name;
	unsigned index;
	float sum;
};
```

За фігурною дужкою, що закриває тіло класу, моє бути крапка з комою. Крапка з комою необхідна, оскільки після тіла класу можливо визначити змінні.

В тілі класу визначені члени (member) класу. В нашому прикладі у класу є лише *змінні-члени* (`data member`). Змінні-члени класу визначають вміст об'єктів цього класу. Кожен об'єкт класу має власний екземпляр змінних-членів класу.

У змінних-членів класу можна визначити *внутрішньокласовий ініціалізатор* (`in-class initializer`). Він використовується для ініціалізації змінних-членів при створенні об'єктів. Члени без ініціалізаторів ініціалізуються за замовчуванням. Внутрішньокласові ініціалізатори мають бути укладені у фігурні дужки або йти за знаком =. Неможна визначити внутрішньокласовий ініціалізатор в круглих дужках.

```
struct MyStruct //Створення власної структури даних (класу)
{
	std::string name; // ініціалізатор за замовчуванням
	unsigned index = 1; // внутрішньокласовий ініціалізатор
	float sum = 1.0; // внутрішньокласовий ініціалізатор
	vector<int> vect{ 1,2,3,4,5 };  //внутрішньокласовий ініціалізатор
	                                // з використанням фігурних дужок
};
```

Мова C++ використовує для визначення власних структур окрім ключового слова `struct` також ключове слово `class`.

Для того, щоб скористатися створеною структурою даних, необхідно визначити об'єкт типу створеної структури. *Визначення об'єкту* типу класу в найпростішому випадку може бути аналогічне визначенню об'єктів інших типів.

```
string word; //визначення змінної вбудованого типу string
```

```
MyStruct my_str; //визначення змінної власного типу MyStruct
```

Після визначення змінної типу класу стає можливим звернення до членів даного класу. Зверення до членів класу використовується для отримання значень змінних-членів, зміни (присвоєння нового значення) значень змінних-членів, виконання певних операцій, які передбачені даним класом. Звернення до членів класу виконується за допомогою *оператору звернення до члену* (`.`).

```
int i = my_str.index; //значення i=1
my_str.sum = 5.5;
vector<int> vec1;
vec1 = my_str.vect; //vec1={ 1,2,3,4,5 }
```

Визначення класу у загальному вигляді з використанням ключового слова `class` виглядає наступним чином:

```
class MyClass //Ім'я класу
{
public:
	//Інтерфейс класу
private:
	//Реалізація класу
};
```

При визначені класу можна використовувати ключові слова `struct` або `class`. Відмінність у визначенні класу за допомогою цих ключових слів полягає у заданому за замовчуванням *рівні доступу*. Якщо використовується *ключове слово* `struct`, то члени, які визначені до першого специфікатора доступу, будуть *відкритими* (`public`), тобто входитимуть до інтерфейсу класу. Якщо використовується *ключове слово* `class`, то члени, які визначені до першого специфікатора доступу, будуть *закритими* (`private`), тобто входитимуть до реалізації класу.

Інкапсуляція надає дві важливі переваги.

- Код користувача не може за необачністю пошкодити інкапсульований об'єкт.
- Реалізація інкапсульованого класу може з часом змінитися і це не вимагатиме змін у коді на рівні користувача.

Змінні, які входять у визначення класу, називають *змінними-членами* (data member).

Члени класу, які є функціями, називають *функціями-членами* (member function).

Функції-члени визначають та об'являють як звичайні функції. Функції-члени *мають бути* об'явлені в класі, проте визначені вони *можуть* бути безпосередньо в класі або поза тілом класу. Функції, які не є членами класу, але є складовими інтерфейсу, об'являються та визначаються поза класом, але в тому ж самому файлі що і клас.

```
class MyClass
{
```

```
public: //Інтерфейс класу
     void Method(); //Об'явлення функції-члену класу без параметрів
private: //Реалізація класу
     float k1 = 1; //Ініціалізація даних-членів класу
     float k2 = 1;
     float k3 = 1;
     float a=-10;
     float b=10;
     double MyFunc(const double&); //Об'явлення функції-члену класу
                                   //з параметром - константним посиланням
     MyClass& MyFunc2(const MyClass&) //Об'явлення функції-члену класу
                 //з параметром - константним посиланням на тип класу
}; //Завершення тіла класу

void MyClass::Method() //Визначення функції-члену класу без параметрів
{                           //поза тілом класу
     double za, zb;
     za = MyFunc(a);
     zb = MyFunc(b);
}

double MyClass::MyFunc(const double &x) //Визначення функції-члену класу
{               //з параметром – константним посиланням поза тілом класу
     double z;
     z = k1*exp(k2*x) + k3*x;
     return z;
}
```

Ім’я функції-члена, яка об’явлена в тілі класу, але визначена поза тілом класу, повинне включати ім’я класу, якому вона належить:

```
void MyClass::Method() //Визначення функції-члену поза тілом класу
```

Ім’я функції `MyClass::Method()` використовує оператор області видимості, щоб вказати, що дана функція об’явлена в межах класу `MyClass`.

Кожен клас визначає власну область видимості. Поза *областю видимості класу* (`class scope`) до звичайних даних і функцій його члени можуть звертатись лише через об’єкт, посилання або покажчик, використовуючи *оператор доступу до члену* (`.`). Для доступу до членів типу з класу використовується *оператор області видимості* (`::`). В любому випадку наступне за оператором ім’я має бути членом відповідного класу.

У функції-члені можна безпосередньо звернутися до членів об’єкту, з якого вона була викликана. Для цього використовується покажчик `this`. *Параметр `this`* визначається неявно та автоматично і його можна використовувати в тілі функції-члену.

```
MyClass& MyClass::MyFunc2(const MyClass &rhs) //Визначення функції-члену
{          //класу з параметром – константним посиланням поза тілом класу
     a += rhs.a;
     return *this; //повертає об’єкт, для якого була викликана функція
```

}

Автори класів інколи визначають *допоміжні функції*, які використовуються як частина інтерфейсу класу, але при цьому членами класу вони не є. Такі функції об'являються (але не визначаються) в тому ж заголовку, що і сам клас після визначення класу. Таким чином, щоб використовувати будь-яку частину інтерфейсу класу, користувачу достатньо підключити лише один файл заголовку.

Кожен клас визначає, як можуть бути ініціалізовані об'єкти його типу. Клас контролює ініціалізацію об'єкту за рахунок визначення однієї чи декількох спеціальних функцій-членів, відомих як *конструктори* (constructor). Задача конструктора – ініціалізувати змінні-члени об'єкту класу. Конструктор виконується кожен раз, коли створюється об'єкт класу.

Ім'я конструктора *співпадає* з іменем класу. На відміну від інших функцій, у конструкторів *немає типу* повернутого значення. Як і інші функції, конструктори мають список параметрів (можливо пустий) і тіло (можливо пусте). У класа може бути *декілька* конструкторів. Подібно будь-якій іншій перевантаженій функції, конструктори мають *відрізнятися* один від одного *кількістю* або *типами* своїх параметрів. Конструктори, на відміну від інших функцій, не можуть бути об'явлені константами.

Якщо в класі не визначено жодного конструктору, класи самі контролюють ініціалізацію відкритих змінних-членів значеннями за замовчуванням, визначаючи спеціальний конструктор, що зветься *стандартним конструктором* (default constructor). Стандартним вважається конструктор, який не отримує ніяких аргументів. В такому випадку змінні-члени ініціалюзуються внутрішньокласовими ініціалізаторами чи значеннями за замовчуванням.

Якщо клас не визначає конструктори явно, компілятор сам визначить стандартний конструктор неявно. Створений компілятором конструктор зветься *синтезованим стандартним конструктором* (synthesized default constructor).

Класи, члени яких мають *вбудований* чи *складений тип* (відповідно не мають ініціалізатора за замовчуванням), можуть розраховувати на синтезований стандартний конструктор, *тільки якщо у всіх* таких членів є внутрішньокласові ініціалізатори.

```
struct MyStruct
{
      int *pi = 0;     //Внутрішньокласовий ініціалізатор змінної
                       //складеного типу
      float y = 1;     //Внутрішньокласовий ініціалізатор змінної
                       //вбудованого типу
      double a;        //Визначення змінної-члену вбудованого типу
                       //без ініціалізатора
}; //Завершення тіла класу
```

Якщо у класа визначений хоча б один конструктор, то компілятор не буде створювати автоматично стандартний конструктор. В такому разі стандартний конструктор потрібно обов'язково визначити в класі самостійно.

Якщо в класі є інші конструктори, то можна попросити компілятор створити *стандартний конструктор* автоматично. Для цього після списку параметрів вказується частина = default.

```
MyClass() = default;
```

Оскільки цей конструктор не має параметрів – він є стандартним конструктором. Застосування цього конструктора можливе лише у випадку, коли для змінних-членів вбудованих та складених типів є *внутрішньокласові ініціалізатори*.

При створенні конструктору класу після переліку параметрів ставиться двокрапка, за якою розміщується *перелік ініціалазації конструктора* (constructor initializer list), який визначає вихідні значення для однієї чи декількох змінних-членів створюваного об'єкту. Завершується конструктор фігурними дужками, які містять тіло конструктора. *Ініціалізатор конструктору* – це перелік імен змінних-членів класу, кожне з яких супроводжується вихідним значенням в круглих (або фігурних) дужках. Якщо ініціалізацій кілька, вони відділяються комами.

```
MyClass() = default; //Стандартний конструктор
MyClass(const float &a1, const float &b1) : a(a1), b(b1) {};
//Конструктор класу
MyClass(const float &kof1, const float &kof2, const int &n, const float
&a1, const float &b1) : k1(kof1), k2(kof2), k3(kof2*n), a(a1), b(b1) {};
//Конструктор класу
```

В даному прикладі першим іде стандартний конструктор, який для ініціалізації змінних членів при створенні об'єкту класу використовує внутрішньокласові ініціалізатори (якщо вони є), в іншому разі використовує ініціалізацію змінних-членів значеннями за замовчуванням (окрім вбудованих та складених типів).

Другим іде конструктор класу, який ініціалізує змінні-члени a та b параметрами a1 та b1 відповідно. Змінні члени k1, k2 та k3 будуть ініціалізовані внутрішньокласовими ініціалізаторами.

Третій конструктор ініціалізує всі змінні-члени створюваного об'єкту класу переліком ініціалізації конструктора, який розміщений між двокрапкою та фігурними дужками. При цьому змінна-член k3 ініціалізується добутком параметрів kof2 та n.

Зазвичай для конструктора краще використовувати внутрішньокласовий ініціалізатор, якщо він є і присвоює члену класу вірні значення. Якщо компілятор не підтримує внутрішньокласову ініціалізацію (введена в стандарті C++11),кожен конструктор повинен явно ініціалізувати кожен член вбудованого типу.

У наведених вище конструкторів тіла пусті. Єдине їх призначення – присвоїти значення змінним-членам. Якщо нічого іншого робити не потрібно, то тіло функції залишається пусти.

Як і інші функції члени, конструктори можуть визначатися як в тілі класу, так і за його межами. В тілі класу визначають функції, які складаються не більше як з одного-двох операторів, щоб не ускладнювати розуміння класу. Функції-члени, в тому числі конструктори, які мають більший обсяг коту у своєму тілі, об'являються в тілі класу, а визначаються одразу після нього в тому ж самому файлі заголовку. Перед іменем конструктора, який визначений за межами класу, як і перед іменем інших функцій-членів, вказується ім'я класу та оператор області видимості.

```
MyClass::MyClass()
{
	//Тіло конструктору
}
```

Ім'я класу перед іменем функції вказується для того, щоб зазначити що дана функція відноситься до вказаного класу. Оскільки ім'я функції співпадає з іменем класу, значить ця функція є конструктором зазначеного класу.

Клас може дозволити іншому класу чи функції отримати доступ до своїх закритих членів, встановивши для них *дружні відносини* (`friend`). Клас об'являє функцію дружньою, включивши її об'явлення з ключовим словом `friend`.

```
class MyClass
{
friend float add(const float&);
friend std::ostream &print(std::ostream&, const MyClass&);
public: //Інтерфейс класу
	MyClass() = default; //Стандартний конструктор
	MyClass(const float &a1, const float &b1) : a(a1), b(b1) {};
	//Конструктор класу
	MyClass(const float &kof1, const float &kof2, const int &n, const
float &a1, const float &b1) : k1(kof1), k2(kof2), k3(kof2*n), a(a1),
b(b1) {}; //Конструктор класу
	void Method(); //Об'явлення функції-члену класу без параметрів
private: //Реалізація класу
	float k1 = 1; //Ініціалізація даних-членів класу
	float k2 = 1;
	float k3 = 1;
	float a=-10;
	float b=10;
	double MyFunc(const double&); //Об'явлення функції-члену класу
                                  //з параметром - константним посиланням
	MyClass& MyFunc2(const MyClass&) //Об'явлення функції-члену класу
                  //з параметром - константним посиланням на тип класу
}; //Завершення тіла класу
//Об'явлення частин, що не є складовими інтерфейсу класу
```

```
float add(const float&);
std::ostream &print(std::ostream&, const MyClass&);
```

Об'явлення друзів може розташовуватись лише *у визначенні* класу, використовуватись в класі вони можуть будь-де. Друзі не є членами класу і не підкоряються специфікатору доступу розділу, в якому вони об'явлені. Об'явлення друзів бажано групувати на початку чи в кінці визначення класу.

Об'явлення дружніх відносин встановлює лише права доступу. Це не об'явлення функції. *Для можливості виклику* дружньої функції користувачами класу її слід також *об'явити*. Для доступу користувачів до дружніх функцій їх зазвичай об'являють поза класом, в тому ж заголовку, що і сам клас.

Класи зазвичай визначають у *файлах заголовку* (header). Файли заголовку мають розширення `*.h`. Як правило, класи зберігаються в заголовках, ім'я яких співпадає з іменем класу. Наприклад, бібліотечний тип `string` визначений у заголовку `string`. За аналогією наш клас `MyStruct` має бути визначений у заголовку `MyStruct.h`.

Заголовки підключаються до програмного файлу за допомогою директиви препроцесора `#include`. Коли препроцесор зустрічає директиву `#include`, він заміняє її вмістом файлу заголовку, який підключений за допомогою цієї директиви. З огляду на це, заголовки також не повинні містити об'явлення `using`, щоб запобігти включенню в програму не передбачених бібліотечних імен.

Заголовки містять сутності, які можуть бути визначені в любому файлі лише раз. Для запобігання кількаразового включення вмісту одного й того самого заголовку, програми C++ використовують препроцесор для *захисту заголовку* (header guard). Захист заголовку опирається на змінні препроцесору. Змінні препроцесору можуть знаходитись у двох станах: визначена чи не визначена. Директива `#define` отримує ім'я і визначає його як змінну препроцесору. Існують дві директиви, що дозволяють перевірити чи була визначена змінна препроцесора. Директива `#ifdef` істинна, якщо змінна була *визначена*, а директива `#ifndef` істинна, якщо змінна *не була визначена*. При істинності перевірки виконується все, що розташоване після директиви `#ifdef` або `#ifndef` і до наступної директиви `#endif`.

Ці засоби можна використовувати для боротьби з багаторазовим включенням вмісту файлів заголовку наступним чином

```
#ifndef MY_STRUCT_H
#define MY_STRUCT_H
#include <string>
struct MyStruct //Створення власної структури даних (класу)
{
    std::string name;
    unsigned index = 1;
```

```
        float sum = 0.0;
};
#endif
```

При першому включенні заголовку MyStruct.h директива #ifndef істинна, і препроцесор обробляє рядки після неї до директиви #endif. В результаті змінна MY_STRUCT_H буде визначена, а вміст заголовку MyStruct.h скопійовано до програми. Якщо надалі включити заголовок MyStruct.h в той же самий файл, то директива #ifndef виявиться брехнею і рядки між нею та директивою #endif будуть проігноровані.

Змінні препроцесора мають бути унікальними в усій програмі. Для цього в ім'я змінної препроцесора включається ім'я класу та ім'я змінної препроцесора задається лише великими літерами. Наприклад змінна препроцесора MY_STRUCT_H для включення визначення класу MyStruct із заголовку MyStruct.h.

Кожен заголовок повинен містити захист від повторного включення до програми його вмісту.

Приклад програмної реалізації

Приклад роботи з класами

```
// Програма пошуку кореня рівняння методом половинного ділення
//

#include "stdafx.h"
#include "MyMethod.h"
#include "windows.h" //Необхідно для виведення в консолі української мови
using std::cout;
using std::cin;
using std::endl;

int main()
{
        SetConsoleCP(1251); //Необхідно для виведення в консолі української мови
        SetConsoleOutputCP(1251); //Необхідно для виведення в консолі української мови
        float k1, k2, k3, a, b, e;
        cout << "Ввведіть коефіцієнти рівняння" << endl;
        cout << "Введіть коефіцієнт k1" << endl;
        cin >> k1;
        cout << "Введіть коефіцієнт k2" << endl;
        cin >> k2;
        cout << "Введіть коефіцієнт k3" << endl;
        cin >> k3;
        cout << "Задайте ліву межу інтервалу пошуку кореня рівняння a" << endl;
        cin >> a;
        cout << "Задайте праву межу інтервалу пошуку кореня рівняння b" << endl;
        cin >> b;
        cout << "Задайте точність пошуку кореня рівняння e" << endl;
        cin >> e;
        MyClass MyCalc(k1, k2, k3, a, b, e);
        MyCalc.Method();
        system("pause"); // Команда затримки екрану
        return 0;
}
```

Приклад створення класу користувача у файлі заголовку

```
// MyMethod.h створений клас для пошуку кореня рівняння методом половинного ділення

#include "math.h"
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

class MyClass
{
public: //Інтерфейс класу
        MyClass() = default; //Конструктор класу за замовчуванням
        MyClass(const float &k1, const float &k2, const float &k3, const float &a1, const float &b1, const
float &e1) : A(a1), B(b1), E(e1), K1(k1), K2(k2), K3(k3) {}; //Конструктор класу
        MyClass(const float &a1, const float &b1) : A(a1), B(b1) {}; //Конструктор класу
        void Method(); //Об'явлення функції-члену класу без параметрів

private: //Реалізація класу
        float K1 = 1; //Ініціалізація даних-членів класу
        float K2 = 1;
        float K3 = 1;
        float A=-100;
        float B=100;
        float E=0.001;
        double MyFunc(const double&); //Об'явлення функції-члену класу з параметром - константним посиланням
};

void MyClass::Method() //Визначення функції-члену класу без параметрів
{
        double za, zb, zx, x;
        double a(A), b(B); //Ініціалізація змінних a та b значеннями  змінних A та B відповідно
        unsigned i=0;
        za = MyFunc(A);
        zb = MyFunc(B);
        cout << "Знаходження коренів рівняння методом половинного ділення" << endl;
        if (za*zb<0)
        {
                do
                {
                        ++i;
                        x = (a + b) / 2;
                        zx = MyFunc(x);
                        cout << "Ітерація " << i << ", x=" << x << ", z=" << zx << endl;
                        za * zx < 0 ? b = x : a = x, za = zx;
                } while (abs(b-a)>E);
                cout << "Корінь рівняння x = " << x << endl;
        }
        else
        {
                cout << "Корінь на інтервалі відсутній. Перевірте відокремлення коренів!" << endl;
        }
}

double MyClass::MyFunc(const double &x) //Визначення функції-члену класу з параметром –
                                        //константним посиланням
{
        double z;
        z = K1*exp(K2*x) + K3*x;
        return z;
}
```

Контрольні питання

1) Що таке власна структура даних або клас? З чого складається визначення класу?
2) Що таке змінні-члени? Що таке внутрішньокласовий ініціалізатор?

3) Що таке абстракція даних? Що таке абстрактний тип даних?
4) Що таке інтерфейс та реалізація класу?
5) Що таке інкапсуляція? Що таке специфікатори доступу?
6) В чому різниця між використанням ключових слів `struct` та `class`?
7) Що таке функції-члени? Де та як вони об'являються та визначаються?
8) Що таке область видимості класу? Наведіть приклад визначення функції-члену поза тілом класу.
9) Для чого використовується параметр `this`? Як звернутися до члену об'єкту класу? Наведіть приклади.
10) Як підключити допоміжні функції для використання класом без їх включення у клас?
11) Що таке конструктор, для чого він використовується?
12) Що таке стандартний конструктор? Що таке синтезований стандартний конструктор? Як об'явити стандартний конструктор?
13) Що таке перелік ініціалізації конструкутору? Для чого він потрібен?
14) Де потрібно визначати конструктор? З чого складається конструктор? Наведіть приклади об'явлення та визначення конструкторів?
15) Що таке дружні відносини? Як об'явити дружню функцію? Як зробити дружню функцію доступною для користувачів класу?
16) Для чого потрібні файли заголовку? Як підключити заголовок до програми? Як організувати захист заголовку?
17) Що таке змінні препроцесора? Які правила створення імен змінних препроцесору та файлів заголовку загальноприйняті? Як працює директива `#include`?

7. Модульна контрольна робота

Для перевірки засвоєння студентами знань, отриманих при прослуховуванні лекцій, виконання комп'ютерних практикумів та при самостійній роботі у відповідності до учбового плану проводиться модульна контрольна робота. Завдання модульної контрольної роботи носять як теоретичний, так і практичний характер. Модульна контрольна робота проводиться за всіма темами кредитного модуля.

Мета: ознайомитись з символьними та чисельними обчисленнями засобами C++; вивчити існуючі типи даних, структуровані типи даних (*вказівники, посилання, створення власної структури даних*); вивчити прийоми роботи з контейнером *vector*; навчитись створювати *файли заголовку*; відпрацювати засоби візуалізації розрахунків в C++; вивчити принципи створення *функцій користувача* та роботи з ними: *визначення функції*; *тіло функції*; тип значення, яке повертає функція з використанням оператору *return*; *об'явлення функцій* у *файлі заголовку*; виклик функції; передача аргументів; створення програм за допомогою функцій.

Завдання: створити консольний додаток для реалізації обчислень поставленої задачі згідно отриманого варіанту завдання з використанням створеної структури даних та покажчиків і посилань та зовнішніх функцій.

Загальні вимоги.

1) Створити консольний додаток з ім'ям виду *PrizvischeMKR*.
2) Програмний код модуля моє бути чітко структурований.
3) Імена об'єктів мають нести сенсові навантаження.
4) Програмний код має супроводжуватись коментарями в тексті програми.

Вимоги до виконання.

1) Створити власну *структуру даних* з ім'ям виду *my_struct*, яка включатиме в якості полів даних змінні для збереження значень *x, y(x), z(x)* та константні змінні *a*, *b* та *c*.
2) Визначити власну *структуру даних* у *файлі заголовку* з ім'ям виду *PrizvischeMKR.h.*
3) Створити відповідні типи *посилань* та *посилань на константу* для кожного поля створеної *структури даних*.

4) Запрограмувати вирішення поставленої задачі з використанням лише *посилань* на поля *структури*.
5) Введення вхідних даних організувати в режимі діалогу з користувачем.
6) Задати початкове, кінцеве значення змінної *x* та крок її зміни.
7) Створити зовнішню функцію для розрахунку *y(x)*.
8) Створити зовнішню функцію для розрахунку *z(x)*.
9) Розрахунок значень функцій *y(x)* та *z(x)* для різних значень змінної *x* організувати у циклі шляхом виклику відповідних зовнішніх функцій на кожній ітерації.
10) Створити вектор типу власної структури даних *my_struct*, та записати в нього значення *x, y(x), z(x)* на кожній ітерації.
11) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеМКР_OUT*.
12) При виведенні вхідних даних та результатів розрахунків використовувати безпосередньо звернення до полів даних створеної *структури* замість *посилань* на них.
13) Виведення даних у вихідних файл організувати в режимі додавання в кінець файлу.

Теоретичні відомості

Вектор (`vector`) – це колекція об'єктів однакового типу, кожному з яких просвоєний цілочисельний індекс, який надає доступ до цього об'єкту. Вектор – це *контейнер* (`container`), оскільки він «містить» інші об'єкти.

Щоб використовувати вектор, необхідно включити відповідний заголовок. Потрібно також включити об'явлення using

```
#include <vector>;
using std::vector;
```

Тип `vector` – це *шаблон класу* (`class template`). Мова C++ підтримує шаблони і класів, і функцій. Шаблони самі по собі не є ні функціями, ні класами. Їх можна вважати інструкцією для компілятора по створенню класів чи функцій. Процес створення компілятором класів чи функцій за шаблоном зветься *створенням екземпляру* (`instantiation`) шаблону. При створенні шаблону необхідно вказати, екземпляр якого класу чи функції має створити компілятор.

Для створення екземрляру шаблону класа слід вказати додаткову інформацію, характер якої залежить від шаблону. Ця інформація завжди задається однаково: в кутових дужках після імені шаблону.

У випадку вектора додатковою інформацією бути тип об'єктів, які він повинен містити:

```
vector<int> vecInt;         //вектор містить об'єкти типу int
vector<string> vecStr;      //вектор містить об'єкти типу string
vector<MyStruct> vecMyStr; //вектор містить об'єкти типу MyStruct,
                            //створеного користувачем
```

Можна визначити вектори для зберігання об’єктів практично любого типу. Оскільки посилання не є об’єктами, не може бути вектору посилань. Також може бути вектор, елементами якого є інший вектор.

```
//Вектор, елементами якого є вектори, що містять текстові рядки
vector<vector<string>> vecVecStr;
```

Подібно будь-якому типу класа, шаблон vector контролює спосіб визначення і ініціалізації векторів. Найбільш розповсюджені способи визначення векторів наведені в табл. 6.1.

Таблиця 7.1 – Способи ініціалізації об’єкту класу `vector`

Ініціалізація	Роз’яснення
`vector<T> v1`	Вектор, що містить об’єкти типу T. Стандартний конструктор, v1 пустий
`vector<T> v2(v1)`	Вектор `v2` - копія всіх елементів вуктору `v1`
`vector<T> v2 = v1`	Еквівалент `v2(v1)`, вектор `v2` - копія елементів вектору `v1`
`vector<T> v3(n, val)`	Вектор `v3` містить `n` елементів зі значенням `val`
`vector<T> v4(n)`	Вектор `v4` містить `n` екземплярів об’єкту типу T, ініціалізованих значенням за замовчуванням
`vector<T> v5{a,b,c …}`	Вектор `v5` містить стільки елементів, скільки вказано у фігурних дужках. Елементи вектору ініціалізуються вказаними у фігурних дужках значеннями (ініціалізація переліком)
`vector<T> v5 = {a,b,c …}`	Еквівалент `v5{a,b,c …}`. Пряма ініціалізація

Ініціалізація вектор за замовчуванням дозволяє створити пустий вектор певного типу.

```
//Ініціалізація за замовчувнннням, у вектора sVec немає елементів
vector<string> sVec;
```

Вектор використовується в мові C++ замість масивів. Шаблон `vector` створений на основі типу масив з додаванням у нього необхідних властивостей. На відміну від масивів, у вектор *можна додавати елементи*. Тому створення пустого вектору певного типу є звичайною операцією з метою подальшого наповнення вектору елементами вказаного типу.

При визначенні вектору для його елементів можна також вказати початкові значення. Наприклад, можна скопіювати елементи з іншого вектора (на відміну від масивів, які копіювати не можна). При копіюванні векторів кожен елемент нового вектору буде копією відповідного елементу вихідного вектору. Обидва вектори повинні мати однаковий тип:

```
vector<int> vecInt(10, 1); //вектор містить десять одиниць
vector<int> iVec1 = vecInt; //вектор iVec1 містить десять одиниць
vector<int> iVec2(iVec1); //вектор iVec2 містить десять одиниць
```

Вектори підтримують *ініціалізацію переліком*, коли значення задаються у фігурних дужках після імені вектору:

```
//Вектор articles міститиме три рядкових елементи a, an та the
vector<string> articles{"a","an","the"};
```

При ініціалізації вектора значення можна пропустити, а вказати тільки кількість елементів. В такому випадку відбудеться *ініціалізація значення* (value initialization), тобто бібліотека створить ініціалізатор елементу самаю. Кількість елементів можна вказувати тільки за допомогою прямої ініціалізації – у круглих дужках після імені вектору:

```
vector<int> iVec(10); //вектор з 10 нулів
vector<string> iVec(10); //вектор з 10 пустих рядків
```

Використання круглих та фігурних дужок дає різні результати:

```
vector<int> vec1(10); //вектор містить 10 елементів зі значенням 0
vector<int> vec2{ 10 }; //вектор містить один елемент зі значенням 10
vector<int> vec3(10,1); //вектор містить 10 елементів зі значенням 1
vector<int> vec4{ 10, 1 }; //вектор містить містить два елементи
                           //зі значенням 10 та 1
```

Зазвичай при створенні вектора невідома кількість його елементів та їх значення. Для додавання елементів у вектор використовують функцію push_back(), яка додає новий елемент в кінець вектора.

```
vector<int> v1; //пустий вектор
for (size_t i = 0; i != 10; i++)
{
    v1.push_back(i + 1); //додавання елементу в кінець вектора зі
                         //значенням i + 1
    cout << v1[i] << endl; //виведення значення вектора на екран
}
```

Результатом роботи даного приклада буде виведення на екран у стовпчик значень створеного вектора, якими будуть числа від 1 до 10.

Окрім функції push_back(), шаблон vector надає ще декілька операцій, більшість з яких потібна операціям класу string. Найбільш важливі з них наведені в табл. 6.2.

Таблиця 7.2 – Операції з векторами

Операція	Роз'яснення
v.push_back(t)	Додає елемент зі значенням t в кінець вектора v
v.empty()	Повертає значення true, якщо вектор v пустий. Інакше повертає значення false
v.size()	Повертає кількість елементів вектору v
v[n]	Повертає посилання на елемент у позиції n вектора v; позиції відраховуються від 0
v1 = {a,b,c …}	Замінює елементи вектору v1 копією елементів із розділеного комами переліку
v1 = v2	Замінює елементи вектору v1 копією елементів вектору v2
v1 == v2	Вектори v1 та v2 рівні, якщо кількість елементів рівна і вони містять однакові елементи на тих самих позиціях
v1 != v2	Вектори v1 та v2 не рівні, якщо хоча б один елемент відрізняється чи різна кількість елементів у векторах
<, <=, >, >=	Мають звичне значення і покладаються на алфавітний порядок символів

Доступ до елементів вектору здійснюється так само, як і до символів у рядку: за їх позицією у векторі.

Функції-члени `empty()` та `size()` працюють аналогічно, як і в класі `string`. Функція-член `size()` повертає значення типу `size_type`, яке визначене відповідним типом шаблону `vector`, наприклад `vector<string>::size_type`.

За допомогою *оператора індексування* можна вибрати вказаний елемент. Подібно рядкам, індексування вектора починається з `0`; индекс має тип `size_type` відповідного типу; якщо вектор не константний, то у повернутий оператором індексування елемент можна здійснити запис; можна розрахувати індекс і безпосередньо звернутись до елементу в даній позиції. За допомогою оператора індексування *неможливо додати елементи* у вектор, для цього використовується лише функція `push_back()`. Оператор індексування дозволяє звертатися (в тому числі і змінювати) лише до вже *існуючих елементів* вектору.

Тип `vector` є контейнером і підтримує роботу з *ітераторами* аналогічно типу `string`. Вектор використовує функції-члени `begin()` та `end()` для звернення до першого та наступного за останнім елементу вектора.

```
vector<float> v{ 0.1, 0.2, 0.3, 0.4, 0.5 }
//Використання ітератора для звернення до елементів вектору
for (vector<float>::iterator iter = v.begin(); iter!= v.end(); iter ++)
{
    //Виведення на екран елементів вектору в рядок
    cout << *iter << " ";
}
```

Оскільки вектор може містити інші вектори, то, наприклад, для збереження елементів матриці використовують вектор векторів. Так для збереження матриці з цілих чисел необхідно створити наступний вектор:

```
vector<vector<int>> vec1;
```

Для спрощення формату запису складних типів використовують псевдоніми типів. *Псевдонім типу* (`type alias`) – це ім'я, яке є синонімом імені типу. Псевдонім типу дозволяє спростити складні визначення типів, полегшуючи їх використання. Псевдоніми типу дозволяють також підкреслити мету використання типу. Визначити псевдонім типу можна одним з двох способів. Традиційно для цього використовують ключове слово `typedef`:

```
typedef double wages; // wages – синонім для double
typedef vector<vector<float>> matrix; // matrix – синонім для двомірного
                                      //вектора (вектора векторів) дійсних чисел
wages w1; //еквівалент об'явлення double w1;
matrix m1; //еквівалент об'явлення vector<vector<float>> m1;
```

Іншим способом створення псевдоніму типу є *об'явлення псевдоніму* (`alias declaration`) за допомогою ключового слова `using` та знаку `=`.

```
using wages = double;
```

```
wages w1;
```

Щоб звернутись до значення вектору можна скористатись ітератором з *оператором звернення до значення* *, наприклад `*iter`. Якщо ми створили вектор векторів, то звернення до значення вектора векторів поверне об'єкт, яким буде вектор.

При зверненні до значення ітератора отримуємо об'єкт, на який вказує ітератор. Якщо цей об'єкт має тип класу, то може знадобитися доступ до члену отриманого об'єкту. Наприклад, якщо є вектор рядків, то може знадобитися дізнатись, чи не пустий певний елемент. З урахуванням того, що `it` – це ітератор даного вектору, можна наступним чином перевірити чи не пустий рядок, на який вказує ітератор:

```
(*it).empty()
```

Круглі дужкі у виразі необхідні, бо вони вимагають застосувати *оператор звернення до значення* (*) до ітератора `it`, а до повернутого ітератором значення застосувати *точковий оператор* (`.`) (оператор звернення до члену класу). Якщо б не було круглих дужок, точковий оператор відносився б до ітератору `it`, а не до повернутого ним об'єкту.

Для спрощення таких виразів, мова пропонує *оператор стрілки* (`->`) (`arrow operator`). Оператор стрілки об'єднує оператори звернення до значення і доступ до члену. Таким чином, вираз `(*it).empty()` можна переписати у спрощеному вигляді:

```
it->empty()
```

Використання векторів разом з ітераторами надає більш гнучкий інструмент для роботи з наборами однотипних елементів, ніж масиви. Всі вектори, на відміну від масивів, є динамічними, дозволяють додавати значення, копіювати та присвоювати вектори, не вимагають програмування звільнення пам'яті.

Функція (`function`) – це іменований блок коду. Запуск цього коду на виконання здійснюється при виклику функції. Функція може отримувати будь-яку кількість аргументів і (зазвичай) повертає результат. Функція може бути перевантажена, відповідно, те ж ім'я може відноситисб до кількох різних функцій.

Визначення функції (`function definition`) зазвичай складається з *типу повернутого значення* (`return type`), імені, переліку *параметрів* (`parameter`) і *тіла функції*. Параметри визначаються в розділеному комами переліку, укладеному в круглі дужки. Дії, які виконує функція, визначаються у блоці операторів, що зветься *тілом функції* (`function body`).

Для запуску коду функції використовується *оператор виклику* (`call operater`), що являє собою пару круглих дужок. Оператор виклику отримує вираз, який є функцією чи покажчиком на функцію. В круглих дужках через кому розміщується перелік *аргументів* (`argument`). Аргументи використовуються для

ініціалізації параметрів функції. Тип викликаного виразу – це тип значення, яке повертає функція.

Для прикладу створимо функцію обчислення факторіалу заданого числа. Факторіал числа n є добутком чисел від 1 до n. Дану функцію можна визначити наступним чином:

```
int fact(int val)
{
	int ret = 1; //локальна змінна для збереження результатів в ході
		//їх обчислення
	while (val>1)
	{
		ret *= val--; // ret = ret*val, val= val-1

	}
	return ret; //повернення результату роботи функції
}
```

Функції присвоєне ім'я `fact`. Вона отримує один параметр типу `int` і повертає значення типу `int`. В циклі `while` обчислюється факторіал з використанням складеного оператора присвоєння та постфіксного оператору декременту. Оператор `return` виконується в кінці функції і повертає значення змінної `ret`.

Для виклику функції `fact()`, потрібно надати їй значення типу `int`. Результатом виклику також буде значення типу `int`.

```
int main()
{
	int j = fact(5); //j = 120, тобто результату fact(5)
	cout << "Факторіал п'яти 5! = " << j << endl;
	return 0;
}
```

Виклик функції виконує дві дії: ініціалізує параметри функції відповідними аргументами і передає керування коду функції. При цьому виконання функції, *яка викликає* (`calling`) призупиняється і починається виконання *викликаної* (`called`) функції.

Виконання функції завершується оператором `return`. Як і виклик функції, оператор `return` виконує дві дії: повертає значення (якщо воно є) і передає керування назад функції, *яка викликає*.

Аргументи – це ініціалізатори для параметрів функції. Перший аргумент ініціалізує перший параметр, другий аргумент ініціалізує другий параметр і т.д. Тип кожного аргументу має співпадати з типом відповідного параметру, як і тип ініціалізатора має співпадати з типом об'єкту, який він ініціалізує. Потрібно передати точно таку ж кількість аргументів, скільку у функції параметрів.

Перелік параметрів функції може бути пустим, проте не може бути відсутнім. При визначенні функції без параметрів зазвичай використовують

пустий список параметрів. Список параметрів, як правило, складається з розділеного комами переліку параметрів, кожен з яких виглядає як одиночне об'явлення. Навіть коли типи двох параметрів однакові, об'явлення слід повторити:

```
int f1(int v1, int v2) {/* ... */}
int f2(int v1, v2) {/* ... */ } //Помилка!!!
```

Параметри не повінні мати однакові імена.

В якості типу повернутого значення функції застосовується більшість типів. Зокрема, типом повернутого значення може бути `void`, це означає, що функція *не повертає значення*. Типом повернутого значення *не може бути* масив чи функція. Проте функція може повертати *покажчик* на масив чи функцію.

В мові C++ ім'я має область видимості, а об'єкт – тривалість існування (`object lifetime`).

- *Область видимості імен* – це *частина тексту програми*, в якій ім'я відомо.
- *Тривалість існування об'єкту* – це *час при виконанні програми*, коли об'єкт існує.

Тіло функції, як блок операторів, формує *нову область видимості*, в якій можна визначати змінні. Параметри та змінні, які визначені в тілі функції, називають *локальними змінними* (`local variable`). Вони є локальними для даної функції і *приховують* (`hide`) об'явлення того ж імені у зовнішній області видимості. Локальні змінні мають пріоритет над глобальними, об'явленими поза межами блоку з тим же ім'ям.

Об'єкти, які визначені поза межами будь-якої з функцій, існують протягом виконання програми. Такі об'єкти створюються при запуску програми і не видаляються до її завершення. Тривалість існування локальної змінної залежить від того, як вона визначена.

Об'єкти, які відповідають звичайним локальним змінним, створюються у місці визначення змінної у функції (блоці) і видаляються, коли процес виконання досягає кінця блоку. Об'єкти, що існують тільки під час виконання блоку,в якому вони були визначені, називають *автоматичними об'єктами* (`automatic object`).

По завершення виконання блоку значення автоматичних об'єктів, які були створені в цьому блоці, невизначені.

Параметри – це автоматичні об'єкти. Значення неініціалізованих локальних змінних вбудованого типу невизначені.

Для створення локальної змінної, тривалість існування якої не переривається між викликами функції, при визначенні локальної змінної використовують ключове слово `static`. Кожен *локальний статичний об'єкт* (`local static object`) ініціалізується раніше, ніж хід виконання досягне

визначення об'єкту. Локальна статична змінна не видаляється по завершенню роботи функції; вона видаляється по завершенню роботи програми.

```
static size_t i = 0; //значення змінної зберігається між викликами функції
```

Як і будь-яке інше ім'я, ім'я функції має бути об'явлене раніше, ніж його можна буде використовувати. Подібно до змінних, функція може бути *визначена* тільки один раз, проте *об'явлена* може бути багаторазово.

Об'явлення функції подібно до її визначення, але у об'явлення *немає тіла функції*. В об'явленні тіло функції замінюється крапкою з комою. Оскільки в об'явленні немає тіла функції, *відпадає необхідність* в іменах параметрів. Проте імена параметрів у об'явленні функції часто використовують для полегшення розуміння користувачем призначення функції.

```
void print(vector<int>::const_iterator beg, vector<int>::const_iterator
end);
```

Три елементи об'явлення функції: тип повернутого значення, ім'я функції та тип параметрів – описують *інтерфейс* (`interface`) функції. Вони задають всю інформація, що необхідна для виклику функції. Об'явлення функції називають також *прототипом функції* (`function prototype`).

Об'явлення функцій розміщають у файлах *заголовку*, а *визначення* – у *файлах вихідного коду*. Файл вихідного коду, в якому функція *визначена*, повинен підключати *заголовок*, в якому функція *об'явлена*. Так компілятор зможе перевірити відповідність визначення та об'явлення.

У *файлі вихідного коду*, де передбачається *використання зовнішньої функції*, також необхідно *підключити заголовок*, в якому ця функція об'явлена. Через файл заголовку з об'явленням функції здійснюється зв'язок між файлом вихідного коду, в якому функція визначена, та файлом вихідного коду, де функція використовується. Відповідно, заголовок з об'явленням функції включається в обидва файли вихідного коду: файл з визначенням функції та файл, в якому вона застосовується. Створені файли заголовку та файли вихідного коду потрібно *підключати* до проекту щоб компілятор їх включив у програму.

Мова C++ дозволяє розділяти програми на логічні частини, надаючі засіб, відомий як *роздільна компіляція* (`separate compilation`). Роздільна компіляція дозволяє поділити програму на кілька файлів, кожен з яких може бути відкомпільований окремо.

При кожному виклику функції її параметри створюються заново. Параметри ініціалізуються так само, як і звичайні змінні. Якщо параметр – посилання, то параметр прив'язується до свого аргументу. В іншому випадку, значення аргументу копіюється.

Коли параметр – посилання, говорять що аргумент *передається за посиланням* (`pass by reference`) або що функція *викликається за посиланням* (`call by reference`). Подібно будь-якому іншому посиланню, параметр

посилання – це лише *псевдонім* об'єкту, до якого він прив'язаний, тобто параметр посилання – це псевдонім свого аргументу.

```
void reset(int i) //i – копія аргументу
{
	i = 0; //змінює значення в середині функції, значення переданого
		//аргументу за межами функції не міняється
}
```

Коли значення аргументу копіюється, параметр і аргумент – незалежні об'єкти. Говорять, що такі аргументи *передаються за значенням* (pass by value) або що функція *викликається за значенням* (call by value).

```
void reset(int &i) //i - лише інше ім'я об'єкту
{
	i = 0; //змінює значення об'єкту, на який посилається і
		//у функції змінюється значення зовнішнього об'єкту
}
```

Функція може повертати лише *одне значення*. У випадках, коли функція має повертати більше ніж одне значення, використовують додаткові *параметри посилання*. Так як посилання – лише інше ім'я об'єкту, то змінюючи такі додаткові параметри всередині функції, ми міняємо значення об'єктів, на які вони посилаються, і за її межами. Тобто повертаємо змінені значення переданих за посиланням аргументів в тіло програми по завершенню виконання функції.

Якщо параметр, який передається у функцію не повинен мінятися, потрібно визначати його як *константне посилання* за допомогою ключового слова const.

```
void print(const double &y)	//Визначення функції для друку значення
				//аргументу
{
	static int i = 0;	//Змінна, яка зберігає своє значення між
				//викликами функції
	cout << "Ітерація " << ++i << endl;
	cout << "y = " << y << endl;
}
```

Використання неконстантних посилань обмежує можливість використання функції. Через константний параметр посилання до функції можна передати як константний, так і неконстантний аргумент. Через неконстантний параметр посилання можна передати лише неконстантний аргумент і не можна передавати константні аргументи, в тому числі рядкові чи чисельні літерали.

Функції, які розташовані в одній області видимості, називають *перевантаженими* (overload), якщо вони мають однакові імена, але різні списки параметрів. При виклику такої функції компілятор приймає рішення про використання певної версії на основі типу переданого аргументу. У перевантажених функцій має однозначно відрізнятися тип параметрів, або має бути різна кількість параметрів.

```
void print(const double &y);
```

```
void print(const int *beg, const int *end);
```

Приклад програмної реалізації

Приклад роботи з функціями користувача

```
//Програма обчислення значень функції F(x,y) в заданих точках

#include "stdafx.h"
#include <iostream>
#include "windows.h"
#include "HeaderFxy.h"          //Підключення власного файлу заголовку, де об'явлена структура даних
#include "fanctFxy.h"           //Підключення власного файлу заголовку, де об'явлена функція f
#include <vector>
using std::vector;
using std::cin;
using std::cout;
using std::endl;


int main()
{
        SetConsoleCP(1251);                     //Необхідно для введення в консолі української мови
        SetConsoleOutputCP(1251);               //Необхідно для виведення в консолі української мови
        while (true)
        {
                const double sgX = 1.9;         //Об’явлення та ініціалізація константної змінної
                const double sgY = 0.2;
                fxy fi;                         //Об’явлення змінної типу створеної структури даних fxy
                vector<fxy> vectF;              //Об’явлення вектору для збереження змінної типу fxy
                size_t n;
                cout << "Задайте початкове значення x(0)" << endl;
                cin >> fi.x;
                cout << "Задайте початкове значення y(0)" << endl;
                cin >> fi.y;
                cout << "Задайте кількість розрахунків значень функції F(x,y)" << endl;
                cin >> n;
                for (size_t i = 0; i < n; i++)
                {
                        functXY(fi);            //Виклик зовнішньої функції розрахунку залежності
                        vectF.push_back(fi);    //Запис змінної типу fxy у вектор
                        printFXY(vectF[i]);     //Виклик зовнішньої функції для друку елементу вектору
                        fi.x += sgX;
                        fi.y += sgY;
                }
                cout << "Запустити розрахунок ще раз? [y] [n]" << endl << endl;
                char y;
                cin >> y;
                if (y == 'n')
                {
                        break;                  //Переривання роботи програми шляхом виходу з циклу while
                }
        }
        return 0;
}
```

Приклад визначення функцій користувача

```
//FunctXY.cpp

#include "stdafx.h"
#include <math.h>
#include <iostream>
#include "HeaderFxy.h"          //Підключення власного файлу заголовку, де об'явлена структура даних
#include "fanctFxy.h"           //Підключення власного файлу заголовку, де об'явлені функції
using std::cin;
using std::cout;
using std::endl;
```

```
void functXY(fxy &f)            //Визначення функції для розрахунку значення на окремій ітерації
{
        f.f = (log(sqrt(f.x)) - log(sqrt(f.y)))*pow((f.x - f.y), 1 / 3) / (1 / tan(f.x));
}

void printFXY(const fxy &f)     //Визначення функції для друку елементів вектору
{
        static int i = 0;       //Змінна, яка зберігає своє значення між викликами функції
        cout << "Ітерація " << ++i << endl;
        cout << "F(" << f.x << "," << f.y << ")=" << f.f << endl;
}
```

Приклад об'явлення функцій користувача у файлі заголовку

```
//functFxy.h


#pragma once
#ifndef FANCTFXY_H              //Попередження повторного визначення класу
#define FANCTFXY_H

void functXY(fxy &f);           //Об'явлення функції functXY з параметром посиланням,
                                //яка не повертає значення

void printFXY(const fxy &f);    //Об'явлення функції printFXY з параметром константне посиланням,
                                //яка не повертає значення

#endif FANCTFXY_H
```

Приклад об'явлення структури даних користувача у файлі заголовку

```
//HeaderFxy.h

#pragma once
#ifndef HEADERFXY_H             //Попередження повторного визначення класу
#define HEADERFXY_H

struct fxy                      //Створення власної структури даних (класу)
{
        double x;
        double y;
        double f;
};

#endif HEADERFXY_H
```

Контрольні питання

1) Що таке вектор? Що потрібно для використання векторів у програмі?
2) Що таке шаблон класу? Що таке створення екземпляру шаблону? Наведіть приклади.
3) Наведіть способи ініціалізації об'єктів класу `vector`.
4) Що таке ініціалізація переліком та ініціалізація значення для об'єкту типу `vector`? В чому відмінність використання круглих та фігурних дужок під час ініціалізації об'єкту типу `vector`, наведіть приклади?
5) Перерахуйте основні операції з векторами.
6) Поясніть як можна отримати доступ до елементів вектора за допомогою індексування та ітераторів? Наведіть приклади.

7) Що таке псевдонім типу та які методи створення псевдонімів ви знаєте? Наведіть приклади об’явлення псевдонімів.
8) Що таке оператор звернення до значення та оператор звернення до члену? Наведіть приклади.
9) Для чого потрібен оператор стрілки? Наведіть приклади.
10) Перерахуйте основні відмінності між масивами та векторами. Який з цих типів більш зручний у використанні?
11) Що таке функція?
12) Що таке визначення функції, тип повернутого значення, параметри, тіло функції?
13) Що таке оператор виклику функції, з чого він складається?
14) Для чого потрібен оператор `return`?
15) Що таке аргументи функції? Для чого використовуються аргументи?
16) Що таке перелік параметрів функції? Як об’являються параметри функції? Чи може бути функція без параметрів? Чи може функція не повертати значення? Наведіть приклади.
17) Що таке область видимості імен та тривалість існування об’єкту?
18) Що таке локальні змінні? Чим вони відрізняються від глобальних?
19) Що таке автоматичний об’єкт? Що таке локальний статичний об’єкт? Наведіть приклади?
20) Що таке об’явлення функції? Що таке інтерфейс функції? Наведіть приклад.
21) Як створити зовнішню функцію? Як підключити зовнішню функцію користувача для використання у файлі вихідного коду?
22) Що таке роздільна компіляція та для чого вона потрібна?
23) Чим відрізняється передача аргументу за посиланням від передачі аргументу за значенням?
24) Скільки значень може повертати функція? Чи може функція повернути кілька значень, якщо так, то як це реалізувати? Наведіть приклади.
25) Як потрібно визначати параметр функції, який не повинен змінюватись? Наведіть приклади.
26) Що таке перевантажені функції? Які особливості їх використання? Наведіть приклади.

Рекомендована література

Базова

1) Стенли Б. Липпман, Жози Лажойе, Барбара Э. Му Язык программирования C++. Базовый курс. М.: Вильямс, 2014. – 1120 с.
2) Бьярне Страуструп Программирование: принципы и практика использования C++. - М.: ООО «И.Д. Вильямс», 2011. – 1248 с.
3) Прата С. Язык программирования C++. Лекции и упражнения. Учебник. - СПб.: ООО «ДиаСофтЮП», 2005. 1104 с.
4) Мейерс С. Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов. - М.: Питер-ДМК, 2006. – 240 с.
5) Прикладне програмне забезпечення - 2. Технології об'єктно-орієнтованого програмування: методичні рекомендації до виконання комп'ютерних практичних робіт для студентів напряму підготовки 6.050202 – «Автоматизація та комп'ютерно-інтегровані технології» [Електронний ресурс] / [уклад. Бендюг В. І., Комариста Б. М.]. – К: 2016. – 153 с.

Допоміжна

6) Аверкин В.П., Бобровский А.И. и др. под ред. Хомоненко А.Д. Программирование на C++. Учебное пособие.Корона-Принт.1999
7) Александреску. Современное проектирование на C++. Обобщенное программирование и прикладные шаблоны проектирования.Вильямс.2002
8) Астахова И.Ф., Власов С.В. Язык C++. Учебное пособие. 2003
9) Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. 2001
10) Глушаков, Коваль, Смирнов. Язык программирования C++, учебный курс. 2001
11) Девис.С.Р. C++ для чайников.Диалектика.2003
12) Дейтел Х, Дейтел П. Как программировать на C++. 1006 с.

Інформаційні ресурси

13) Язык программирования C++ [Електрон. ресурс] // Основы программирования на языках Си и C++ для начинающих - Режим доступу: http://cppstudio.com/cat/274/
14) Бендюг Владислав Іванович [Електрон. ресурс] // Офіційний сайт – Режим доступу: http://бендюг.укр/
15) Єдине інформаційне середовище - Національний технічний університет України «КПІ» [Електрон. ресурс] // Електронний КАМПУС НТУУ «КПІ» – Режим доступу: http://login.kpi.ua/

Додаток А

Титульний аркуш

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Хіміко-технологічний факультет

Кафедра кібернетики хіміко-технологічних процесів

Комп'ютерний практикуму №1

з дисципліни

«ТЕХНОЛОГІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ»

тема: Створення консольного додатку

Виконав: ст. групи ХА-41 Іваненко І.І.
Перевірив: доц. каф. КХТП Бендюг В.І.

Оцінка, балів:
Виконання _____ ;
оформлення звіту _____ ;
захист результатів _____.

Загальна оцінка _____.

_____
(підпис викладача)

Київ – 2016

Індивідуальні завдання до домашньої контрольної роботи

1. Обчислення кореня заданого нелінійного рівняння $f(x) = 0$ на попередньо визначеному інтервалі $[a, b]$[1] **методом хорд** з заданою точністю ε.
2. Обчислення кореня заданого нелінійного рівняння $f(x) = 0$ на попередньо визначеному інтервалі $[a, b]$[1] **методом дотичних** з заданою точністю ε.
3. Обчислення кореня заданого нелінійного рівняння $f(x) = 0$ на попередньо визначеному інтервалі $[a, b]$[1] **комбінованим методом хорд-дотичних** з заданою точністю ε.
4. Розв'язання системи лінійних алгебраїчних рівнянь **методом ітерацій** з заданою точністю ε.
5. Розв'язання системи лінійних алгебраїчних рівнянь **методом Гауса-Зейделя** з заданою точністю ε.
6. Розв'язання системи двох нелінійних рівнянь **методом ітерацій** з заданою точністю ε.
7. Розв'язання системи двох нелінійних рівнянь **методом Ньютона** з заданою точністю ε.
8. Розв'язання задачі інтерполяції (як прямої, так і оберненої) **методом Лагранжа**.
9. Розв'язання прямої задачі інтерполяції (екстраполяції) **методом Ньютона**.
10. Розв'язання оберненої задачі інтерполяції (екстраполяції) **методом Ньютона**.
11. Розв'язання звичайного диференціального рівняння першого порядку **методом Ейлера**.
12. Розв'язання звичайного диференціального рівняння першого порядку **удосконаленим методом Ейлера**.
13. Розв'язання звичайного диференціального рівняння першого порядку **методом Ейлера-Коші**.
14. Розв'язання звичайного диференціального рівняння першого порядку **методом Ейлера-Коші з наступною ітераційною обробкою**.
15. Розв'язання звичайного диференціального рівняння першого порядку **методом Рунге-Кутта.**
16. Обчислення визначеного інтегралу **методом трапецій**.
17. Обчислення визначеного інтегралу **методом Сімпсона**.
18. Розв'язання системи лінійних алгебраїчних рівнянь **методом ітерацій** з заданою точністю ε.
19. Розв'язання системи лінійних алгебраїчних рівнянь **методом Гауса-Зейделя** з заданою точністю ε.
20. Розв'язання системи лінійних алгебраїчних рівнянь 2-го порядку **за правилом Крамера**.

21. Розв’язання системи лінійних алгебраїчних рівнянь 3-го порядку **за правилом Крамера**.
22. Розв’язання задачі екстраполяції (як прямої, так і оберненої) **методом Лагранжа**.

Індивідуальні завдання до модульної контрольної роботи

1. $y(x) = 2 - x - \ln x,$ $z(x) = \sqrt[3]{3ax^2 + bx} + \tan(cx - 8).$

2. $y(x) = 0{,}3e^{-0{,}6x} - x,$ $z(x) = cx^3 - a\cos x^2 + b.$

3. $y(x) = 2^x + 4x - 3,$ $z(x) = \frac{x^3+a}{bx-1} + \sqrt[3]{3x^2} + cx.$

4. $y(x) = e^{-2x} - 3x + 1,$ $z(x) = \frac{ax^3-3x^2}{bx-2} + 3c\sin x.$

5. $y(x) = \ln(4{,}6x) - 5{,}2x + 1{,}4,$ $z(x) = ax^3\sin(b + x) - c\,x.$

6. $y(x) = 0{,}4\,e^{-0{,}7x} - x - 6,$ $z(x) = \mathrm{a}\tan x^2 - (bx^2 + c\,x)/(x + 4).$

7. $y(x) = \ln 3x - 3{,}4x + 2{,}6,$ $z(x) = \frac{ax^3-3x^2}{bx-5} + \cos cx.$

8. $y(x) = 0{,}81\,e^{-0{,}6x} - 2x,$ $z(x) = \mathrm{a}\log x^3 - (bx^2 + cx)/4b.$

9. $y(x) = \ln(1{,}5x) - 1{,}7x + 3,$ $z(x) = x^3 - ax^2 + \tan(bx - c).$

10. $y(x) = 0{,}6\,e^{-0{,}5x} - 0{,}8x,$ $z(x) = \sqrt[3]{x^3 - ax^2} + \sin(bx - c).$

11. $y(x) = 3^x + 2x - \ln x,$ $z(x) = x^3 - \sqrt{ax^2 + 2bx} - \cos cx.$

12. $y(x) = 2e^x - 3x + 1,$ $z(x) = \log_a x^3 + b\sqrt[3]{cx^2 - 8}.$

13. $y(x) = \ln 4x - 4{,}5x + 2,$ $z(x) = (x^2 + ax)^{3/2} - \frac{2-bx}{3x^2+cx} + 3b.$

14. $y(x) = 0{,}73e^{-0{,}6x} - x,$ $z(x) = \frac{1}{x^3-ax^2} + \sqrt{3bx - c}.$

15. $y(x) = \ln 6x - 7{,}1x + 1,$ $z(x) = 3a\cos^2(-3bx^2) + \frac{bx-2}{cx}.$

16. $y(x) = 0{,}9e^{-0{,}8x} - 0{,}5x,$ $z(x) = x^{3a}\sin\left(\frac{ax}{b+3x} - c\right) - bx^2.$

17. $y(x) = \lg(7{,}6x) - 8{,}6x + 0{,}5,$ $z(x) = \frac{ax^3-5x^2}{b+\tan(cx^2)} + bx - c.$

18. $y(x) = 2e^x - 2x + 3,$ $z(x) = \sqrt[3]{3ax^2 + bx} + \cos(cx - 8b).$

19. $y(x) = \lg(8{,}5x) - 9{,}6x + 2,$ $z(x) = \frac{1}{x^3} - 5ax^{2-bx} + \sin(cx - 5b).$

20. $y(x) = e^{-3x} - 3x + 1,$ $z(x) = \mathrm{a}\tan^2 3x^2 - \frac{bx^2+cx}{ax-5b} - c.$

21. $y(x) = e^{-3x} - 3x + 1$, $z(x) = \sqrt[3]{ax^3 + 3bx^2} + \frac{\sin cx}{2bx+1} + 3a$.

22. $y(x) = \lg 2{,}3x - 2{,}5x + 2{,}7$, $z(x) = (x^3 - ax^2)^{bx} + \cos(cx + 2) - b$.

23. $y(x) = 0{,}54e^{-0{,}56x} - 0{,}98x$, $z(x) = (ax^3 + x^2)^{1/bx} + c\cos x + bx$.

24. $y(x) = \lg 3{,}8x - 4{,}3x + 2$, $z(x) = \frac{2ax+\sin(bx-3)}{x^3-cx^2} + bx$.

25. $y(x) = 0{,}65e^{-0{,}6x} - 0{,}89x$, $z(x) = a\cos 2x + x^3 - bx^2 + c\sin(bx - 5)$.

26. $y(x) = \lg 0{,}9x - 1{,}4x + 2{,}8$, $z(x) = \tan^2 x^3 - \frac{ax^2+bx}{cx+3} + b$.

27. $y(x) = 0{,}47e^{-0{,}7x} - 0{,}93x$, $z(x) = \sqrt[3]{\frac{x^3-ax^2}{bx^3}} + \sin(cx - b)$.

28. $y(x) = 5^x + 4x^{2-x} - 3$, $z(x) = \frac{\cos(ax+b)}{\sqrt{cx^3}+2x}x^3 + bx$.

29. $y(x) = \lg 3{,}8x - 4{,}6x + 2{,}1$, $z(x) = 3a\tan x^3 - \sqrt{3x^2 + bx} - c$.

30. $y(x) = 0{,}17\, e^{-0{,}86x} - 1{,}1x$, $z(x) = \frac{x^3-3ax^2}{\sin 2x^3} + \sqrt{bx - c}$.