# ТЕХНОЛОГІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

# МЕТОДИЧНІ ВКАЗІВКИ
## ДО ВИКОНАННЯ КОМП'ЮТЕРНИХ ПРАКТИКУМІВ

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

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

# МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ КОМП'ЮТЕРНИХ ПРАКТИКУМІВ

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 РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ КОМП'ЮТЕРНИХ ПРАКТИКУМІВ 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. Систем. вимоги: Intel Core 2; 1 Gb RAM; Windows 7; Adobe Acrobat – Назва з екрану.

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

Електронне навчальне видання

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

# МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ КОМП'ЮТЕРНИХ ПРАКТИКУМІВ

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

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 С++;

- 💻 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.

## Структура кредитного модуля

### Лекційні заняття

| № з/п | Назва теми лекції та перелік основних питань |
|---|---|
| | ***Розділ 1 Введення в C++*** |
| **1** | *Тема 1.1. Створення простої програми на C++.* Створення консольного додатку Win32. Компіляція та запуск.<br>*Тема 1.2. Операції введення-виведення та коментарі*. Стандартні оператори введення-виведення. Введення даних з файлу. Виведення інформації у файл. Використання коментарів.<br>*Література: [*1, с. 25-36*].* |
| **2** | *Тема 1.3. Засоби керування ходом виконання програми.* Оператор while. Оператор for. Введене невідомої кількості даних. Оператор if.<br>*Тема 1.4. Введення в класи.* Поняття класу. Перший погляд на функції-члени.<br>*Література: [*1, с. 37-51*].* |
| | ***Розділ 2 Основи мови об'єктно-орієнтованого програмування C++*** |
| **3** | *Тема 2.1. Змінні та базові типи.* 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.<br>*Література: [*1, с. 61-118*].* |
| **4** | *Тема 2.2. Типи string, vector та масиви.* Побудова імен і визначення using. Бібліотечний тип string: визначення та ініціалізація рядків; операції з рядками; робота з символами рядка. Бібліотечний тип vector: визначення та ініціалізація 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 C; взаємодія із застарілим кодом; багатомірні масиви.<br>*Література: [*1, с. 123-182*].* |

| | |
|---|---|
| **5** | *Тема 2.3. Вирази.* 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.<br>*Література: [*1, с. 187-228*].* |
| **6** | *Тема 2.4. Оператори.* 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.<br>*Література: [*1, с. 233-263*].* |
| **7** | *Тема 2.5. Функції.* 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.<br>*Література: [*1, с. 267-325*].* |
| **8-9** | *Тема 2.6. Класи.* 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. Область видимості класу. Застосування конструкторів: список ініціалізації конструктору; делегуючий конструктор; роль стандартного конструктора; неявне перетворення типів класу; агрегатні класи; літеральні класи; статичні члени класу.<br>*Література: [*1, с. 331-391*].* |

## Комп’ютерний практикум

Комп'ютерній практикум проводиться з метою закріплення знань, одержаних при читанні лекцій та самостійній роботі та вироблення вмінь і досвіду створення програмних додатків з використанням мови програмування С++.

| № з/п | Назва комп’ютерного практикуму | Кількість ауд. годин |
|---|---|---|
| **1** | *Створення консольного додатку.* Загальні навички роботи в командному режимі. Компіляція та запуск. Оператори введення-виведення. Коментарі. | 6 |
| **2** | *Оператор if та логічні оператори.* Використання математичних функцій в С++, підключення бібліотеки математичних функцій. Застосування оператора if. Логічні оператори та оператори відношення. | 6 |
| **3** | *Засоби керування процесом виконання програми.* Ітераційний оператор while. Оператор арифметичного циклу for. | 6 |
| **4** | *Змінні та базові типи.* Символьні та чисельні обчислення засобами С++, типи даних, покажчики, посилання, створення власної структури даних, файли заголовку. Засоби візуалізації розрахунків в С++. | 6 |
| **5** | *Типи string та масиви.* Бібліотечний тип string: операція з рядками; робота з символами рядка. Використання оператора індексування для типу string та масиву. Використання ітераторів для типу string. Масиви: визначення та ініціалізація; доступ до елементів; застосування покажчиків. | 6 |
| **6** | *Тип vector.* Бібліотечний тип vector: визначення і ініціалізація; додавання елементів; інші операції. Використання ітераторів для типу vector. Використання двомірних векторів. | 6 |
| **7** | *Оператори.* Умовні оператори: оператор if, оператор switch. Ітераційні оператори: цикл do while, серійний цикл for. Оператори переходу: break, continue. Блоки try та обробка виключень: оператор throw, блок try. | 6 |
| **8** | *Функції.* Принципи створення функцій користувача та роботи з ними: визначення функції; тіло функції; тип значення, яке | 6 |

| | | |
|---|---|---|
| | повертає функція з використанням оператору return; об'явлення функцій у файлі заголовку; виклик функції; передача аргументів; створення програм за допомогою функцій. | |
| **9** | *Класи.* Рішення систем диференційних рівнянь з використанням засобів програмування С++. | 6 |

## Індивідуальні завдання

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.

## Контрольні роботи

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.

## 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.

1. Комп'ютерні практичні роботи.

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.

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.

3. Домашня контрольна робота

Ваговий бал – 20.

Домашня контрольна робота являє собою практичне завдання – створення програмного модулю на мові програмування C++ згідно отриманого варіанту індивідуального завдання. За результатами виконання роботи для позитивної оцінки студент може отримати від 13 до 20 балів.

# 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.

***Загальні вимоги.***

1) Створити консольний додаток з ім'ям виду `PrizvischeKP1`.
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 новий проект `Visual C++` типу `Win32 Console Application`.
2) Розробити алгоритм розрахунку залежності згідно свого варіанту.
3) Підключити бібліотеку *потоків введення-виведення* `iostream`.
4) Організувати введення вхідних даних за допомогою об'єкту `cin` стандартної бібліотеки `std`.
5) Організувати виведення запитів користувачу та результатів розрахунку за допомогою об'єкту `cout` стандартної бібліотеки `std`.
6) Вставити в тексті програми *коментарі* для пояснення її роботи.
7) Встановити в тексті програми *точку зупинки* і в *покроковому режимі* переглянути поточні значення змінних.

### Теоретичні відомості

Кожна програма C++ містить одну чи декілька *функцій* (`function`), при цьому одна з них обов'язково має ім'я `main()`. Запускаючи програму C++ операційна система викликає саме функцію `main()`.

Визначення функції містить чотири елементи: *тип повереного значення* (`return type`), *ім'я функції* (`function name`), *список параметрів* (`parameter list`), який може бути пустий, і *тіло функції* (`function body`).

```
тип поверненого значення ім'я функції(список параметрів)
{
```

```
    Тіло функції;
}
```

Функція `main()` повинна мати тип поверненого значення `int`, який є типом цілих чисел. Тип `int` – це *вбудований тип* (`built-in type`) даних, такі типи визначені у самій мові.

Заключна частина визначення функції, її тіло, являє собою блок операторів (`block of statements`), який починається відкриваючою *фігурною дужкою* (`curly brace`) та завершується закриваючою фігурною дужкою.

```
int main()
{
    return 0;
}
```

Оператор `return` завершує код функції. Він може передати значення назад стороні, яка викликала функцію. Значення, яке отримує оператор return, за типом має сумісним з типом поверненого значення функції.

Більшість операторів мови C++ завершуються крапкою з комою. Її відсутність призводить до виведенні компілятором повідомлень про помилки.

В більшості операційних систем повернене функцією `main()` значення використовується як індикатор стану. Повернене значення `0` свідчить про успіх. Будь-яке інше значення, як правило, означає відмову, а саме значення вказує на її причину.

Файли програм зазвичай називають *файлами вихідного коду* (`source file`). Ім'я файлу завершується суфіксом (розширенням). Різні компілятори використовують суфікси файлів вихідного коду; до найбільш поширених відносять `.cc`, `.cxx`, `.cpp`, `.cp` та `.C`.

В самій мові C++ немає операторів для *введення та виведення* (`Input/Output - IO`). Їх функції забезпечує *стандартна бібліотека* (`standard library`). Основу *бібліотеки* `iostream` складають два типи, `istream` та `ostream`, які представляють потоки введення та виведення відповідно. *Потік* (`stream`) – це послідовність символів, яка записується чи зчитується з пристрою введення-виведення деяким способом. Термін «потік» передбачає, що символи надходять та передаються послідовно протягом певного часу.

В бібліотеці визначені чотири об'єкти введення-виведення. Для здійснення введення використовують об'єкт `cin` (вимовляється «сі-ін») типу `istream`. Цей об'єкт називають також *стандартним введенням* (`standard input`). Для виведення використовується об'єкт `cout` (вимовляється «сі-аут») типу `ostream`. Його часто згадують як *стандартне виведення* (`standard output`).

В бібліотеці визначені ще два об'єкти типу `ostream` – це `cerr` та `clog` (вимовляється як «сі-ерр» та «сі-лог» відповідно). Об'єкт `cerr` називають також *стандартною помилкою* (`standard error`) та, як правило, використовують в

програмах для створення попереджень і повідомлень про помилки, а об'єкт clog – для створення інформаційних повідомлень.

Для включення компілятором в програму бібліотеки iostream використовується *директива препроцесору* (preprocessor directive)

```
#include <iostream>
```

Ім'я в кутових дужках – це *заголовок* (header). Кожна програма, яка використовує засоби, що зберігаються в бібліотеці, має підключити відповідний заголовок. *Директива* #include має бути написана в одному рядку та знаходитись поза тілом функції. Зазвичай всі директиви #include програми розташовані на початку файлу вихідного коду.

Перший оператор в тілі функції main() виконує *вираз* (expression). В мові C++ вираз складається з одного чи кількох *операндів* (operand) і, як правило, *оператора* (operator). Щоб відобразити підказку на стандартному пристрої виведення, використовується *оператор виведення* (output operator), або *оператор* <<.

```
std::cout << "Input t1" << std::endl;
```

В даному рядку об'єднані два оператори виведення. Перший оператор std::cout << "Input t1" виводить повідомлення для користувача. Це повідомлення є *рядковим літералом* (string literal) – послідовність символів, які заключені у подвійні лапки. Оператор складається з двох операндів, лівий повинен бути об'єктом класу ostream, а правий операнд – це значення для відображення. Оператор << заносить передане значення в об'єкт cout класу ostream. Таким чином, результатом буде об'єкт класу ostream, в який записане передане значення. Текст у подвійних лапках виводиться на стандартний пристрій виведення.

Другий оператор std::cout << std::endl виводить endl (читається як «енд-ел») – спеціальне значення, що зветься *маніпулятором* (manipulator). При його запису у потік виведення здійснюється перехід на новий рядок та скидання *буферу* (buffer), який пов'язаний з даним пристроєм. Скидання буферу гарантує, що все виведення, яке програма сформувала на даний момент, буде одразу записане в потік виведення, а не буде очікувати запису, знаходячись у пам'яті.

Оскільки перша частина оператору повертає об'єкт класу ostream, то два оператори

```
std::cout << "Input v1 and v2";
std::cout << std::endl;
```

можна об'єднати, що і було зроблено.

В наведених операторах використана форма запису std::cout та std::endl замість cout та endl. Префікс std:: означає, що імена cout та endl визначені у *просторі імен* (namespace) на ім'я std. Простори імен дозволяють

уникнути імовірних конфліктів, причинами яких може стати співпадіння імен, визначених в різних бібліотеках. Всі імена, які визначені в стандартній бібліотеці, знаходяться у просторі імен `std`. В запису `std::cout` застосовується *оператор області видимості* `::` (`scope operator`), який вказує що тут використовується ім'я `cout`, яке визначене в просторі імен `std`.

Після запиту на введення даних, необхідно організувати читання введених користувачем даних. Але перед цим потрібно визначити *змінні* (`variable`), в які будуть запам'ятовуватись введені значення.

```
int v1 = 0, v2=0;
```

Ці змінні визначені як такі, що відносяться до типу `int`. Тип `int` є вбудованим типом даних для цілочисельних значень. Окрім того, ми також *ініціалізуємо* (`initialize`) змінні, тобто присвоюємо вказане значення на момент створення.

*Оператор введення* `>>` (`input operator`) поводить себе аналогічно оператору виведення. Його лівим операндом (відносно оператору `>>`) є об'єкт класу `istream`, а правим операндом – об'єкт, який запам'ятовує отримані дані. Подібно оператору виведення, оператор введення повертає як результат свій лівий операнд (об'єкт класу `istream`), тому можна об'єднувати кілька операторів введення

```
std::cin >> v1 >> v2;
```

що рівноцінно

```
std::cin >> v1;
std::cin >> v2;
```

## Приклад програмної реалізації

***Приклад створення консольного додатку, використання операторів введення-виведення, додавання коментарів.***

```
//Приклад створення консольного додатку та використання операторів введення-виведення
//

#include "stdafx.h"
#include <iostream>

int main() //Розрахувати об'єм газу за залежністю P1*V1/T1=P2*V2/T2
{
	/*При заданих значеннях температури в цельсіях, тиску в кПа та об'єму в л газу
	розрухувати об'єм газу при іншій температурі та тиску*/
	float t1, p1, v1, t2, p2, v2;			//Об'явлення змінних дійсного типу, які
використовуються в розрахунках
	const int t0 = 273;				//Об'явлення цілочисельної константи для переведення
температури з градусів Цельсія в Кельвіни
	std::cout << "Input t1" << std::endl;		//Виведення на екран запиту для введення температури
	std::cin >> t1;					//Запис введеного з клавіатури значення в змінну t1
	std::cout << "Input P1" << std::endl;		//Виведення на екран запиту для введення тиску
	std::cin >> p1;					//Запис введеного з клавіатури значення в змінну p1
	std::cout << "Input V1" << std::endl;		//Виведення на екран запиту для введення об'єму газу
	std::cin >> v1;					//Запис введеного з клавіатури значення в змінну v1
	std::cout << "Input t2" << std::endl;		//Виведення на екран запиту для введення температури,
при якій необхідно розрахувати об'єм газу
	std::cin >> t2;					//Запис введеного з клавіатури значення в змінну t2
	std::cout << "Input P2" << std::endl;		//Виведення на екран запиту для введення тиску, при
якому необхідно розрахувати об'єм газу
	std::cin >> p2;					//Запис введеного з клавіатури значення в змінну p2
	v2 = p1*v1*(t2 + t0) / (p2*(t1 + t0));		//Розрахункова залежність для об'єму газу
	std::cout << "V2" << v2 << std::endl;		//Виведення на екран розрахованого об'єму газу
	return 0;					//Завершення роботи функції main() і повернення
значення 0
}
```

## Контрольні питання

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 С++, 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.

***Загальні вимоги.***

1) Створити консольний додаток з ім'ям виду *PrizvischeKP2*.
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) Розробити алгоритм і програму для визначення наведеної залежності при певних значеннях вхідної змінної.
2) Вибір залежності для розрахунку вихідної змінної організувати з використанням *умовного оператору if*.
3) Виведення результатів розрахунку організувати з вказуванням вхідних значень змінних, при яких був отриманий результат, а також вказуванням варіанту розрахункової залежності, за якою було отримане значення вихідної змінної.
4) Підібрати значення вхідних змінних таким чином, щоб отримати значення вихідної змінної за кожним варіантом розрахункової залежності.

### Теоретичні відомості

Подібно іншим мовам прграмування С++ містить *оператор* `if`, який забезпечує виконання операторів за умовою. Структуру оператора можна представити наступним чином:

```
if (true) //умова виконання блоку операторів
{
	//блок операторів 1
}
else //якщо умова не виконана
{
	//блок операторів 2
```

```
}
```

Існують дві форми оператора `if`: з розділом `else` та без нього. В обох версіях умова береться у круглі дужки. *Умова* може бути виразом або ініціалізуючим об'явленням змінної. Тип виразу або змінної повинен дозволяти перетворення в тип `bool`. Оператор `if` – керуючий оператор, який забезпечує виконання на основі значення певної умови. Якщо умова істинна (значення `true`), виконується тіло оператора `if`. В іншому випадку (значення `false`) управління переходить до розділу `else` (якщо він існує).

Існують *унарні оператори* (`unary operator`) та *парні оператори* (`binary operator`). Унарні оператори, такі як *оператор звернення до адреси* `&` та *оператор звернення до значення* `*`, діють на один операнд. Парні чи бінарні оператори, такі як рівність (==) та множення (*), діють на два операнди. Деякі символи, наприклад *, використовуються для позначення як унарних (звернення до значення), так і парних (множення) операторів. Тип оператору визначає контекст, в якому він використовується. У використанні цих операторів немає ніякого зв'язку, тому їх слід вважати двома різними символами.

Таблиця 2.1 – Арифметичні оператори

| Оператор | Дія | Застосування |
|---|---|---|
| + | унарний плюс | + вираз |
| - | унарний мінус | - вираз |
| * | множення | вираз * вираз |
| / | ділення | вираз / вираз |
| % | залишок | вираз % вираз |
| + | додавання | вираз + вираз |
| - | віднімання | вираз - вираз |

В табл. 2.1 оператори згруповані за пріоритетом. Унарні арифметичні оператори мають більш високий пріоритет, ніж оператори множення та ділення, які в свою чергу мають більш високий пріоритет, ніж парні оператори віднімання та додавання.

Таблиця 2.2 – Логічні оператори та оператори відношення

| Оператор | Дія | Застосування |
|---|---|---|
| ! | логічне NOT | ! вираз |
| < | менше | вираз < вираз |
| <= | менше чи дорівнює | вираз <= вираз |
| > | більше | вираз > вираз |
| >= | більше чи дорівнює | вираз >= вираз |
| == | рівність | вираз == вираз |
| != | не дорівнює | вираз != вираз |
| && | логічне AND | вираз && вираз |
| \|\| | логічне OR | вираз \|\| вираз |

Загальним результатом *оператора логічного AND* (&&) буде `true`, якщо обидва його операнди розглядаються як `true`. *Оператор логічного OR* (||) повертає значення `true`, якщо будь-який з його операндів розглядається як `true`.

*Оператор логічного NOT* (`!`) повертає обернене значення свого операнда (`!true` повертає `false`).

В якості *оператора присвоєння* (`assignment operator`) в C++ використовується оператор `=` (наприклад, `a=5`) на відміну від логічного *оператора рівності* (`equality operator`) `==` (перевірка рівності лівої та правої частини виразу в умові), який використовується в логічних виразах (наприклад, `if (a==5) ++i;`).

В C++ також використовують *складені оператори присвоєння* (`compound assignment`), які існують для кожного з арифметичних операторів.

Таблиця 2.3 – Складені оператори присвоєння

| Оператор | Застосування | Приклад |
|---|---|---|
| *= | вираз *= вираз | a *= b рівноцінно a = a * b |
| /= | вираз /= вираз | a /= b рівноцінно a = a / b |
| %= | вираз %= вираз | a %= b рівноцінно a = a % b |
| += | вираз += вираз | a += b рівноцінно a = a + b |
| -= | вираз -= вираз | a -= b рівноцінно a = a - b |

В C++ існують *оператор приростання ++* (`increment`) та *оператор зменшення* `--` (`decrement`), які дозволяють в короткій та зручній формі додавати чи віднімати одиницю з об'єкту. Ця форма запису часто використовується при роботі з *ітераторами*, оскільки більшість ітераторів не підтримують арифметичні дії.

Ці оператори існують в двох формах: префіксній (`prefix`) та постфіксній (`postfix`). *Префіксний оператор приростання* (*зменшення*) здійснює збільшення (зменшення) свого операнду на одиницю і повертає *змінений* об'єкт як результат. *Постфіксний оператор приростання* (*зменшення*) повертає копію вихідного операнду *незмінною*, а потім здійснює збільшення (зменшення) свого операнду на одиницю.

```
int i = 0, j;
j = ++i;
std::cout << "j=" << j << ", i=" << i << std::endl;
// j=1, i=1 префікс повертає збільшене значення
int k = 0, l;
l = k++;
std::cout << "l=" << l << ", k=" << k << std::endl;
// l=0, k=1 постфікс повертає вихідне значення
```

До сих пір імена зі стандартної бібліотеки згадувались у програмах явно, тобто перед кожним з них було вказано ім'я простору імен `std`. Наприклад, при читанні зі стандартного пристрою введення застосовувалась форма запису

std::cin. Тут застосований оператор області видимості ::. При частому використанні бібліотечних імен така форма запису може бути надто громіздкою. Щоб уникнути цього застосовують об'явлення using (using declaretion). *Об'явлення using* дозволяє використовувати імена з іншого простору імен без вказування префіксу *ім'я_простору_імен*::. Об'явлення using має наступний формат:

```
using простірімен::ім'я;
```

Після того як об'явлення using було зроблено один раз, до вказаного в ньому імені можна звертатися без вказування простору імен.

```
#include <iostream>
using std::cout;
using std::endl;
int main()
{
	int i = 0, j;
	j = ++i;
	cout << "j=" << j << ", i=" << i << endl;
	return 0;
}
```

Для кожного імені необхідно індивідуальне об'явлення using. Об'явлення using не можна застосовувати у файлах заголовку для запобігання конфлікту імен.

Щоб використовувати у програмі математичні функції необхідно приєднати до програми заголовковий файл math.h за допомогою директиви препроцесора #include.

```
#include <math.h>
```

Таблиця 2.4 – Основні математичні функції

| Функція | Пояснення |
|---|---|
| **abs(x)** | модуль цілого числа |
| **fabs(x)** | модуль дробового числа |
| **sin(x)** | синус |
| **cos(x)** | косинус |
| **tan(x)** | тангенс |
| **asin(x)** | арксинус |
| **acos(x)** | арккосинус |
| **atan(x)** | арктангенс |
| **log(x)** | натуральний логарифм |
| **log10(x)** | десятковий логарифм |
| **exp(x)** | піднесення e до ступеню x |
| **pow(x,y)** | піднесення x до ступеню y |
| **pow10(x)** | піднесення 10 до ступеню x |
| **sqrt(x)** | квадратний корінь |
| **ceil(x)** | округлення вгору |

| Функція | Пояснення |
|---|---|
| `floor(x)` | округлення вниз |
| `fmod(x,y)` | остача від ділення x на y |

## Приклад програмної реалізації

***Приклад роботи з умовним оператором if та математичними функціями.***

```
// ConsoleApplication44.cpp : Програма розрахунку вихідної змінної в залежності від значення вхідної з використанням умовного оператора.
//

#include "stdafx.h"
#include <windows.h> //Необхідно для виведення в консолі української мови
#include <math.h> //Бібліотека математичних функцій
#include <iostream>
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
	double t, x, z;
	unsigned i; //беззнакове ціле число
	cout << "Задайте значення вихідної змінної t" << endl;
	cin >> t;
	x = exp(-sin(t)) + cos(t);
	if (x<=-3)
	{
		z = sqrt(abs(x))/(t+x);
		i = 1;
	}
	else if (x > -3 && x<2)
	{
		z = t+tan(pow(x,3));
		i = 2;
	}
	else if (x==2)
	{
		z = sin(pow(x, 3));
		i = 3;
	}
	else
	{
		z = log(x + t);
		i = 4;
	}
	cout << "При t=" << t << " значення x=" << x << ", розрахунок проведений за залежністю " << i << ", z=" << z << endl;
	return 0;
}
```

## Контрольні питання

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*.

***Загальні вимоги.***

1) Створити консольний додаток з ім'ям виду *PrizvischeKP3*.
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.***

***Частина 1.***

1) Створити програму для розрахунку шуканих значень результуючої величини в заданому діапазоні значень вхідної змінної.
2) При програмуванні використовувати *арифметичний цикл for*.
3) Вхідні константні значення об'являти у вигляді *константних змінних* певного типу.
4) Виведення результатів розрахунку організувати у табличному вигляді з виведенням вхідної та вихідної змінної на кожній ітерації розрахунку.

***Частина 2.***

1) Створити програму для розрахунку наближеного значення функції за допомогою суми нескінченого ряду.
2) Організувати перевірку належності введеного значення змінної *x* вказаному інтервалу за допомогою *умовного оператора if*.
3) Організувати розрахунок суми нескінченного ряду з використанням *ітераційного оператору while*. Умовою виходу з циклу має бути точність обчислень ряду.

4) Передбачити виведення на екран значень суми на кожній ітерації розрахунку. Вивести на екран контрольне значення розрахованої функції.

## Теоретичні відомості

*Оператор* `while` організовує ітераційне (циклічне) виконання фрагменту коду, поки його умова залишається істинною. Структура оператору `while` виглядає наступним чином

```
while (true) //умова
{
	//блок операторів
}
```

Оператор `while` (оператор ітераційного циклу) циклічно виконує *блок операторів*, доки *умова* залишається істинною. *Умова* – це вираз, результатом виконання якого є істина (`true`) чи брехня (`false`). Допоки *умова* істинна (`true`), *блок операторів* виконується. Після виконання *блоку операторів умова* перевіряється знову. Якщо *умова* залишається істинною, *блок операторів* виконується знову. Цикл `while` продовжується, почергово перевіряючи *умову* та виконуючи *блок операторів*, допоки *умова* не стане брехнею (`false`).

*Блок* (`block`) – це послідовність з будь-якої кількості операторів, що заключено у фігурні дужки. Блок є оператором і може використовуватись будь-де, де припустимий один оператор.

```
while (val<=10) //цикл виконується допоки значення val не перевищить 10
{
	sum += val;//присвоїти sum суму sum та val
	++val; //додати 1 до val
}
```

*Оператор* `for` (оператор арифметичного циклу) організовує задану кількість повторень блоку операторів. У кожного оператора `for` є дві частини: *заголовок* та *тіло*. Заголовок контролює кількість разів виконання *тіла циклу*. Сам заголовок складається з трьох частин: *оператора ініціалізації*, *умови* і *виразу*.

```
for (оператор ініціалізації; умова; вираз) //заголовок циклу
{
	//тіло циклу
}
```

Оператор ініціалізації визначає змінну циклу та задає її початкове значення. В умові міститься перевірка досягнення змінною циклу кінцевого значення. У виразі здійснюється прирощення змінної циклу.

```
for (int i = 0; i <= 10; i++) //заголовок циклу
{ //тіло циклу
	sum += val;
	cout << "Сума = " << sum << endl;
}
```

В даному прикладі в *операторі ініціалізації* визначається *змінна циклу* `i` цілочисельного типу `int` та ініціалізується значенням `0`. В *умові* перевіряється чи не досягла *змінна циклу* кінцевого значення `10`. У *виразі* вказується прирощення *змінної циклу*, яке дорівнює `1`. Якщо після перевірки *умови змінна циклу* не досягла свого кінцевого значення, виконується *тіло циклу*. Після виконання *тіла циклу* відбувається прирощення *змінної циклу* на `1` і знову перевіряється *умова* досягнення *змінною циклу* кінцевого значення.

В операторі арифметичного циклу `for` на відміну від оператору ітераційного циклу `while` заздалегідь відома кількість *ітерацій* – повторень *тіла циклу*. Вона дорівнює

$$\text{кількість ітерацій} = \frac{\begin{matrix}\text{кінцеве значення}\\ \text{змінної циклу}\end{matrix} - \begin{matrix}\text{початкове значення}\\ \text{змінної циклу}\end{matrix}}{\begin{matrix}\text{прирощення}\\ \text{змінної циклу}\end{matrix}}$$

Для наведеного прикладу кількість ітерацій циклу дорівнюватиме `(10-0)/1=10`, тобто перед початком роботи оператора `for` відомо, що цикл виконається `10` разів.

Програми часто потребують джерело випадкових чисел. До стандарту C++ 11 для цього використовувалась проста бібліотечна функція мови C на ім'я `rand()`. Дана функція створює псевдовипадкові цілі числа, які рівномірно розподілені в діапазоні від `0` до залежного від системи максимального значення, котре не менше 32767. Але більшості програм потрібні випадкові числа в зовсім іншому діапазоні. Деяким додаткам потрібні випадкові числа з плаваючою комою, або числа з неоднорідним розподілом.

Ці проблеми вирішує бібліотека випадкових чисел, яка визначена у заголовку random. Дана бібліотека включає набір взаємодіючих класів: класів *процесора випадкових чисел* (`random-number engine`) та класів *розподілу випадкового числа* (`random-number distribution`).

Процесор створює послідовність беззнакових випадкових чисел, а розподіл використовує процесор для створення випадкових чисел визначеного типу в заданому діапазоні, який розподілений згідно вказаного імовірнісного розподілу.

В C++ використовують клас `default_random_engine` для генерування простих випадкових чисел, але ці числа зазвичай лежать у відмінному від потрібного діапазону. Для отримання числа в певному діапазоні, використовують об'єкт типу розподілу.

```
#include <random> //бібліотека генерування випадкових чисел
int main()
{
	//Створює випадкове беззнакове ціле число
	std::default_random_engine e;
	//Повертає однорідно розподілене ціле число в заданому діапазоні
```

```
	std::uniform_int_distribution<int> u(-5, 5);
	//цикл виводить 10 чисел з заданого діапазону
	for (int i = 0; i < 10; i++)
	{
		//випадкове ціле число з заданого діапазону
		std::cout << u(e) << " ";
	}
	return 0;
}
```

Результатом роботи програми буде наступний рядок чисел

```
1 -2 5 4 -1 2 -5 1 0 -1
```

Тут u визначається як об'єкт типу `uniform_int_distribution<int>`. Цей тип створює однорідно розподілені значення. При визначенні об'єкту даного типу можна задати мінімум та максимум необхідних значень. Визначення `u(-5, 5)` вказує, що необхідні числа в діапазоні від -5 до 5 включно. Розподіл випадкового числа використовує включений діапазон, який дозволяє отримати будь-яке можливе цілочисельне значення в ньому.

Для отримання набору випадкових чисел з плаваючою комою необхідно визначити об'єкт типу `uniform_real_distribution<int>`. Подібно до типу `uniform_int_distribution<int>`, тут також можна задати мінімальне та максимальне значення при визначенні об'єкту

```
#include <random> //бібліотека генерування випадкових чисел
int main()
{
	//Створює випадкове беззнакове ціле число
	std::default_random_engine e;
	//Повертає однорідно розподілене дійсне число в заданому діапазоні
	std:: uniform_real_distribution<double> u(-5, 5);
	//цикл виводить 5 чисел з заданого діапазону
	for (int i = 0; i < 5; i++)
	{
		//випадкове ціле число з заданого діапазону
		std::cout << u(e) << " ";
	}
	return 0;
}
```

Результатом роботи програми буде наступний рядок чисел

```
-3.64523 3.35009 4.68868 -2.78966 -1.91833
```

В обох попередніх прикладах при кожному запуску програми буде генеруватись та ж сама послідовність випадкових чисел. Для отримання кожного разу нового набору випадкових чисел можна скористатись системною функцією `time()`, яка визначена у заголовку `ctime`. Функція `time()` повертає кількість секунд, починаючи з заданої епохи. Дана функція отримує один параметр, який

є покажчиком на структуру для запису часу. Якщо цей покажчик нульовий, функція лише повертає час

```
default_random_engine e(time(0)); //майже випадкове початкове число
```

З використанням функції `time()` попередній приклад набуде наступного вигляду

```
#include <random> //бібліотека генерування випадкових чисел
#include <ctime> //бібліотека, яка містить функцію time()
int main()
{
	//Створює випадкове беззнакове ціле число
	std::default_random_engine e(time(0));
	//Повертає однорідно розподілене дійсне число в заданому діапазоні
	std:: uniform_real_distribution<double> u(-5, 5);
	//цикл виводить 5 чисел з заданого діапазону
	for (int i = 0; i < 5; i++)
	{
		//випадкове ціле число з заданого діапазону
		std::cout << u(e) << " ";
	}
	return 0;
}
```

Результатом роботи програми буде наступний рядок чисел

```
0.912443 -4.218 -3.05783 4.38754 4.08929
```

При новому запуску програми набір чисел буде інший

```
1.49806 -0.775143 -4.46224 -4.58333 0.434274
```

## Приклад програмної реалізації

***Приклад роботи з умовним оператором if та оператором ітераційного циклу while (Частина 1).***

```
//Обчислення наближеного значення функції sin(pi/x)
//

#include "stdafx.h"
#include <iostream>
#include <windows.h> //Необхідно для виведення в консолі української мови
#include <math.h> //Бібліотека математичних функцій
using std::cout;
using std::cin;
using std::endl;


int main() //Розрахунок суми нескінченого ряду
{
	SetConsoleCP(1251); //Необхідно для виведення в консолі української мови
	SetConsoleOutputCP(1251); //Необхідно для виведення в консолі української мови
	double x, s = 0, s1=1, sk, p=2, fact=1;
	const double eps = 0.0001, pi=3.14;
	const int a = 180;
	int n = 0;
	cout << "Ввведіть значення X" << endl;
	cin >> x;
	//double s = 1 / x;
	if (abs(x)<=a && abs(x)>0) //розруховуємо значення суми ряду для x з певного діапазону 0<|x|<=a
	{
		sk = sin(pi/x); //значення контрольної суми
		while (abs(s - s1)>eps) //перевірка точності обчислень суми ряду
		{
			s1 = s;
			fact *= (2 * n + 1); //розрахунок факторіалу
			s += pow(-1, n)*(pow(pi, (2*n+1)))/(pow(x, (2 * n + 1))*fact);
			cout << "Ітерація " << n+1 << " поточне значення суми нескінченного ряду S=" << s <<
endl;
			n++;
		}
	}
	else
	{
		cout << "Значення має бути в межах 0<|x|<="<< a << endl;
	}

	cout << "Контрольна сума sin(pi/x)= " << sk << endl;
	return 0;
}
```

***Приклад роботи з умовним оператором if та оператором арифметичного циклу for, використання генератора випадкових чисел (Частина 2).***

```
//Обчислення суми чисел в заданому діапазоні, підрахунок від’ємних чисел серед випадково згенерованих
//

#include "stdafx.h"
#include <iostream>
#include <random> //Бібліотека генерування випадкових чисел
#include <ctime> //Бібліотека, яка містить функцію time()
#include <windows.h> //Бібліотека, яка 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
	cout << "Введіть значення x" << endl;
	int x,y,lower,upper,sum=0;
	cin >> x;
	cout << "Введіть значення y" << endl;
	cin >> y;
	if (x>y)
	{
		upper = x;
		lower = y;
		cout << "x > y, x = " << x << endl;
	}
	else
	{
		upper = y;
		lower = x;
		cout << "x <= y, y = " << y << endl;
	}
	for (int i = lower; i <= upper; i++) //цикл рахує суму чисел від меньшого до більшого
	{
		sum += i;
		cout << "sum = " << sum << endl;
	}
	std::default_random_engine e(time(0)); //Створює випадкове беззнакове ціле число
	std::uniform_int_distribution<int> u(-lower, upper); //Повертає однорідно розподілене значення в
заданому діапазоні
	int n,z,neg=0;
	cout << "Введіть кількість чисел для пошуку серед них від'ємних значень" << endl;
	cin >> n;
	for (int i = 0; i < n; i++) //цикл рахує кількість від'ємних чисел
	{
		z=u(e); //випадкове ціле число з заданого діапазону
		cout << z << " ";
		if (z<0)
		{
			neg++;
		}
	}
	cout << endl;
	cout << "Кількість від'ємних значень у наведеному вище рядку = " << neg << endl;
	return 0;
}
```

## Контрольні питання

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.

***Загальні вимоги.***

1) Створити консольний додаток з ім'ям виду *PrizvischeKP4*.
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.***

***Частина 1.***

1) Створити власну *структуру даних* з ім'ям виду *PrizvischeKP4_z1*, яка включатиме в якості полів даних всі необхідні вхідні та вихідні параметри задачі з обов'язковою наявністю *цілочисельних, дійсних та рядкових типів*, а також *константних змінних*.
2) Визначити власну *структуру даних* у *файлі заголовку* з ім'ям виду *PrizvischeKP_z1.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) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП4_з1_OUT*.

7) При виведенні вхідних даних та результатів розрахунків використовувати безпосередньо звернення до полів даних створеної *структури* замість *посилань* на них.
8) Виведення даних у вихідних файл організувати в режимі додавання в кінець файлу.

***Частина 2.***

1) Створити власну *структуру даних* з ім'ям виду *PrizvischeKP4_z2*, яка включатиме в якості полів даних всі необхідні вхідні та вихідні параметри задачі з обов'язковою наявністю *цілочисельних, дійсних* та *рядкових типів*, а також *константних змінних*.
2) Визначити власну *структуру даних* у *файлі заголовку* з ім'ям виду *PrizvischeKP4_z2.h.*
3) Створити відповідні типи *покажчиків* та *покажчиків на константу* для кожного поля створеної *структури даних*.
4) Запрограмувати вирішення поставленої задачі з використанням лише *покажчиків* на поля *структури*.
5) Введення вхідних даних організувати в режимі діалогу з користувачем.
6) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП4_з2_OUT*.
7) При виведенні вхідних даних та результатів розрахунків використовувати безпосередньо звернення до полів даних створеної *структури* замість *покажчиків* на них.
8) Виведення даних у вихідний файл організувати в режимі заміни вмісту файлу.

## Теоретичні відомості

В мові C++ визначений набір базових типів, включно з *арифметичними типами* (`arithmetic type`) та спеціальним типом `void`. *Арифметичні типи* представляють символи, цілі числа, логічні значення та числа з плаваючою крапкою. З типом `void` не пов'язано значень і застосовується він тільки за деяких обставин, частіше за все як тип значення, що повертає функція, котра не повертає нічого.

Існує два різновиди арифметичних типів: *цілочисельні типи* (включно з символьними і логічними) та *типи з плаваючою крапкою*. Розмір (тобто кількість бітів) арифметичних типів залежить від конкретного комп'ютера. Стандарт гарантує мінімальні розміри, наведені в табл. 4.1.

Таблиця 4.1 – Арифметичні типи мови C++

| Тип даних | Значення | Мінімальний розмір |
|---|---|---|
| `bool` | Логічний тип | Не визначений |
| `char` | Символ | 8 бітів |
| `wchar_t` | Широкий символ | 16 бітів |
| `char16_t` | Символ Unicode | 16 бітів |
| `char32_t` | Символ Unicode | 32 біти |
| `short` | Коротке ціле число | 16 бітів |
| `int` | Ціле число | 16 бітів |
| `long` | Довге ціле число | 32 бітів |
| `long long` | Довге ціле число | 64 бітів |
| `float` | Число з плаваючою комою одинарної точності | 6 значущих цифр |
| `double` | Число з плаваючою комою подвійної точності | 10 значущих цифр |
| `long double` | Число з плаваючою комою підвищеної точності | 10 значущих цифр |

Тип `bool` приймає лише значення true (істина) та false (брехня).

Існує кілька *символьних типів*, більшість з яких призначена для підтримки національних наборів символів. *Типи з плаваючою крапкою* надають значення з одинарною, подвійною та розширеною точністю. Стандарт визначає мінімальну кількість значущих цифр.

За виключенням типу `bool` і розширених символьних типів *цілочисельні типи* можуть бути *знаковими* (`signed`) або *беззнаковими* (`unsigned`). *Знаковий тип* може представляти від'ємні та додатні числа (включно з нулем), а *беззнаковий тип* – тільки додатні числа та нуль.

Типи `int`, `short`, `long` та `long long` є знаковими. Відповідні беззнакові типи отримують приставку `unsigned` до назви типу, наприклад `unsigned long`. Тип `unsigned int` може бути скорочений до `unsigned`.

Тип об'єкту визначає дані, які він може містити, та операції, які з ним можна виконувати. Серед операцій, які підтримуються множиною типів, є можливість *перетворювати* (`convert`) об'єкти даного типу в інший, пов'язаний тип.

Перетворення типів відбувається автоматично, коли об'єкт одного типу використовується там, де очікується об'єкт іншого типу. Коли значення одного арифметичного типу присвоюється іншому, результат залежить від діапазону значень, які підтримує тип.

- Коли значення одного з не логічних арифметичних типів присвоюється типу `bool`, результатом буде `false`, якщо значення є `0`, а в іншому разі – `true`.
- Коли значення типу `bool` присвоюється одному з інших арифметичних типів, буде отримане значення `1`, якщо логічним значенням було `true`, та `0`, якщо значення було `false`.

- Якщо значення з плаваючою крапкою присвоюється об’єкту цілочисельного типу, воно урізується до частини перед десятковою крапкою.
- Коли цілочисельне (інтегральне) значення присвоюється об’єкту типу з плаваючою крапкою, дробова частина дорівнює нулю. Якщо у цілого числа більше бітів, ніж може вмістити об’єкт з плаваючою крапкою, то точність може бути *втрачена*.
- Якщо об’єкту беззнакового типу присвоюється значення не з його діапазону, результатом буде залишок від ділення за модулем значення, яке може містити тип призначення. Тобто присвоєне значення буде зовсім *іншим*.
- Якщо об’єкту знакового типу присвоюється значення не з його діапазону, результат буде *невизначеним*.

*Змінна* (variable) –це іменоване сховище, яким можуть маніпулювати програми. У кожної змінної в мові C++ є тип. Тип визначає розмір і розташування змінної в пам’яті, діапазон значень, які можуть зберігатись у ній, та набір операцій, які можна застосувати до змінної. Програмісти C++ використовують терміни «змінна» та «об’єкт» як синоніми.

Просте визначення змінної складається зі *специфікатора типу* (type specifier), який супроводжується переліком із одного чи декількох імен змінних, що відділені комами, та завершується крапкою з комою.

```
unsigned char c;
int i, t=0, k;
bool b;
float f, f1;
```

*Ініціалізація* (initialization) присвоює об’єкту певне значення у момент створення. Значення, які використовуються для ініціалізації змінних, можуть бути як завгодно складними виразами.

```
//змінна price визначена і ініціалізована раніше, ніж вона використана для
//ініціалізації змінної discount
double price = 99.99, discount = price*0.15;
```

При визначенні змінної без ініціалізатора відбувається її *ініціалізація за замовчуванням* (default initialization). Таким змінним присвоюється *значення за замовчуванням* (default value). Це значення залежить від типу змінної і може також залежати від того, де визначається змінна.

Значення об’єкту вбудованого типу, яке не ініціалізовано явно, залежить від того, де саме воно визначається. Змінні, які визначені поза межами функції, ініціалізуються значенням 0. Значення *неініціалізованих* (uninitialized) об’єктів вбудованого типу, які визначені у тілі функції, невизначені.

Рекомендується ініціалізувати кожен об'єкт вбудованого типу. Це не завжди необхідно, але простіше, ніж з'ясовувати, чи можна в даному конкретному випадку обійтись без ініціалізатора.

*Ідентифікатори* (identifier) або імена в мові C++ можуть складатись з символів, цифр та символів підкреслення. Мова не накладає обмежень на довжину імен. Ідентифікатори мають починатися з літери чи символу підкреслення. Символи у верхньому та нижньому регістрі відрізняються, тобто ідентифікатори мови чутливі до регістру.

Існує певний набір ключових слів, зарезервованих в мові C++, наприклад auto, int, class, enum, struct, new та деякі інші. Дані слова не можуть використовуватись у якості ідентифікаторів. Також імена змінних не повинні починатись *з двох символів підкреслення* підряд, наприклад __indx не можна використовувати в якості ідентифікатора. Крім того ім'я змінної не може починатися *з символу підкреслення за яким іде прописна літера*, наприклад _Indx заборонено використовувати в якості імені об'єкту.

Існують певні загальноприйняті правила для іменування змінних. Дотримування цих правил покращує зручність читання коду.

- Ідентифікатор має бути змістовним.
- Імена змінних мають містити лише рядкові символи. Наприклад index, а не Index чи INDEX.
- Імена *класів* зазвичай починаються з прописної літери, наприклад Birthday_struct.
- Декілька слів в ідентифікаторі розділяють символами підкреслення або прописними першими літерами кожного слова окрім першого символу ідентифікатора, наприклад student_group або studentGroup, але не studentgroup.

*Область видимості* (scoup) –це частина програми, в якій у імені є конкретне значення. Як правило, області видимості в мові C++ розмежовуються фігурними дужками. Імена видимі від моменту їх об'явлення і до кінця області видимості, в якій вони об'явлені.

*Складений тип* (compound type) –це тип, який визначений в термінах іншого типу. У мові C++ є декілька складених типів, два з яких це посилання та покажчики.

*Посилання* (reference) є альтернативним іменем об'єкту. Тип посилання «посилається на» інший тип. У визначенні типу посилання використовується *оператор об'явлення* у формі &d, де d – ім'я, що об'являється. Символ & слід повторювати для кожного імені покажчика.

```
int &a, &b, c; //c не є посиланням
int i;
int &j = i;
```

Зазвичай при ініціалізації змінної значення ініціалізатора копіюється в створюваний об'єкт. При визначенні посилання замість копіювання значення ініціалізатора відбувається *зв'язування* (bind) посилання з його ініціалізатором. Після ініціалізації посилання залишається зв'язаним з вихідним об'єктом. Неможливо змінити прив'язку посилання і зв'язати його з іншим об'єктом, тому посилання *слід* ініціалізувати.

Посилання – це не об'єкт, а тільки *інше ім'я (псевдонім) вже існуючого об'єкту*.

- Після того, як посилання визначене, *всі* операції з ним фактично здійснюються з об'єктом, з яким пов'язане посилання.
- При присвоєнні посилання присвоюється об'єкт, з яким воно пов'язане.
- При доступі до значення посилання фактично відбувається звернення до значення об'єкту, з яким пов'язане посилання.
- Коли посилання використовується як ініціалізатор, в дійсності для цього використовується об'єкт, з яким пов'язане посилання.
- Оскільки посилання не об'єкти, неможливо визначити посилання на посилання.
- Типи посилання та об'єкту, на який воно посилається мають співпадати точно.
- Посилання може бути пов'язане лише з об'єктом, але не з літералом чи результатом більш загального виразу.

```
int i;
float &k = i; //Помилка! Неспівпадіння типів посилання та об'єкту
int &j = 10; //Помилка! Посилання на літерал
```

*Покажчик* (pointer) – це складений тип, змінна якого вказує на об'єкт іншого типу. Подібно посиланням, покажчики використовують для опосередкованого доступу до інших об'єктів.

- На відміну від посилання, покажчик – це справжній *об'єкт*.
- Покажчики можуть бути присвоєні та скопійовані.
- Один покажчик за час свого існування може вказувати на кілька різних об'єктів.
- На відміну від посилання, покажчик можна не ініціалізувати у момент визначення.
- Типи покажчика та об'єкту, яким він ініціалізується мають співпадати.
- Оскільки посилання не об'єкти, в них немає адрес, а відповідно, неможливо визначити покажчик на посилання.

Тип покажчика визначається оператором у формі *d, де d – ім'я покажчика, що створюється. Символ * слід повторювати для кожної змінної покажчика.

```
int *p1, *p2, c; //c не є покажчиком
```

Покажчик містить адресу іншого об'єкту. Для отримання адреси об'єкту використовують *оператор звернення до адреси* (address-of operator), або оператор &.

```
int i=77;
int *p=&i; //p містить адресу змінної i, p - покажчик на змінну i
```

У прикладі p визначене як покажчик на тип int та ініціалізоване адресою об'єкту i типу int.

Коли покажчик вказує на об'єкт, для доступу до цього об'єкту можна використати *оператор звернення до значення* (dereference operator), або *оператор* *.

```
int i=28;
int *p=&i; //p містить адресу змінної i, p - покажчик на змінну i
cout << *p; //* повертає об'єкт, на який вказує p, виводиться 28
```

Звернення до значення покажчика (*) повертає об'єкт, на який вказує покажчик.

Коли складно при об'явленні об'єкту визначити тип його можливих значень, можна скористатися *специфікатором типу auto*. На відміну від інших специфікаторів типів (int, double), які задають певний тип, специфікатор auto наказує компілятору вивести тип з ініціалізатора. У зв'язку з цим, у змінної що використовує специфікатор auto, має бути ініціалізатор.

```
//item ініціалізується результатом суми val1 + val2
auto item = val1 + val2;
//специфікатор auto може визначати кілька змінних
auto i = 0, *p = &i;
```

Для визначення змінної, тип якої компілятор виводить з виразу, але не використовує цей вираз для ініціалізації змінної, застосовують *специфікатор типу decltype*. Специфікатор decltype повертає тип його операнду.

```
decltype(f()) sum = x;
```

Тут компілятор не викликає функцію f(), але він використовує тип, який повернув би такій виклик для змінної sum. Таким чином, компілятор призначає змінній sum той же тип, якій був би повернутий при виклику функції f().

В мові C++ можна створювати власні типи даних, створюючи клас. 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, яке буде розглянуте пізніше.

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 }
```

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.

При введенні чи виведенні в консольному додатку *символів кирилиці* виникне помилка або вони будуть некоректно відображені на екрані. При програмуванні під Windows використовуються наступні *таблиці кодування*: cp866, cp1251 и utf-8 (стандарт Unicode). Хоча вже давно розроблений єдиний стандарт кодування символів - Unicode, в Windows досі використовуються кілька кодіровочних таблиць, а саме - cp866 та cp1251. Використання декількох таблиць кодування символів і є причиною появи некоректних символів, замість повідомлення українською мовою.

Unicode - єдиний стандарт кодування символів, що дозволяє представити знаки всіх письмових мов. Таким чином *стандарт Unicode* привласнює кожному символу унікальний код, незалежно від мови. Зараз Unicode вважається кращим стандартом кодування символів.

Так уже повелося, що в командному рядку Windows кодування символів відповідає стандарту cp866. Тобто всі символи в командному рядку Windows закодовані по кодувальній таблиці cp866. Причому поміняти кодування в командному рядку Windows неможна.

У всіх україномовних Windows кодування cp1251 є стандартним 8 - бітним кодуванням. І при створенні проекту в Microsoft Visual Studio цей стандарт кодування символів успадковується проектом, тобто програмою. Хоча кодування для проекту в Visual Studio можна легко поміняти, це не вирішує проблеми, оскільки консоль розуміє тільки одне кодування cp866, якого в MVS немає. У результаті, виходить, що програма передає коди символів повідомлення за стандартом cp1251. Командний рядок приймає ці коди і переводить їх у символи, але вже за стандартом cp866, оскільки іншого стандарту не знає. У підсумку повідомлення було надіслане консолі, але символи інтерпретовані неправильно, ось так і з'являються некоректні символи.

Для того, щоб запити в консольному додатку відображалися українською мовою необхідно підключити до проекту файл заголовку windows.h. У файлі містяться прототипи функцій SetConsoleCP() і SetConsoleOutputCP(), вони нам і потрібні. Аргументом цих функцій є ідентифікатор кодової сторінки, потрібна нам сторінка win-cp 1251. Функція SetConsoleCP() встановлює потрібну кодову таблицю на потік введення, тоді як функція SetConsoleOutputCP() встановлює потрібну кодову таблицю на потік виводу.

В результаті до нашої програми потрібно додати три додаткові рядки:

```
#include "windows.h"
SetConsoleCP(1251);
```

```
SetConsoleOutputCP(1251);
```

Приклад програми з коректним використанням кирилиці буде наступним:

```
#include "stdafx.h"
#include <iostream>
#include "windows.h"  //Необхідно для використання функцій SetConsoleCP()
                      //та SetConsoleOutputCP()
using std::cout;
using std::cin;
using std::endl;
int main()
{
      SetConsoleCP(1251);         //Необхідно для введення в консолі
                                  //української мови
      SetConsoleOutputCP(1251);   //Необхідно для виведення в консолі
                                  //української мови
      char ch = 'т'; //ініціалізація змінної ch початковим значенням 'т'
      while (ch != 'н') //умова виходу з циклу
      {
            cout << "Введіть два значення:";
            int val1 = 0, val2 = 0;
            cin >> val1 >> val2; //значення вводяться через пробіл
            cout << "Сума " << val1 << " та " << val2 << " = "
                 << val1 + val2 << "\n"
                 << "Продовжити введення? Введіть 'т' чи 'н': ";
            cin >> ch;
      }
      return 0;
}
```

Даний приклад сумує будь-яку кількість пари чисел за допомогою циклу `while`. Умова циклу `while` перевіряє, який символ увів користувач у відповідь на запит `"Продовжити введення? Введіть 'т' чи 'н': "`. Якщо користувач введе будь-який символ окрім ‘н’, цикл продовжить своє виконання, тобто програма запросить ввести наступні два числа для розрахунку їх суми. При введенні користувачем у відповідь на запит `"Продовжити введення? Введіть 'т' чи 'н': "` символу ‘н’, виконання циклу завершиться.

## Приклад програмної реалізації

***Приклад роботи зі структурами та функціями.***

```
//Визначення знаку зодіаку людини за датою її народження

#include "stdafx.h"
#include <string>
#include <iostream>
#include "windows.h"
#include " BirthdayStruct.h"
#include "Zodiac.h"
using std::cin;
using std::cout;
using std::endl;
using std::string;

int _tmain(int argc, _TCHAR* argv[]) //Створення власної структури даних та функції користувача з підключенням їх
{       //через файли заголовку, використання посилань
        SetConsoleCP(1251); //Необхідно для виведення в консолі української мови
        SetConsoleOutputCP(1251); //Необхідно для виведення в консолі української мови
        BirthdayStruct Birthday; //Створення змінною типу власної структури (класу) BirthdayStruct
        cout << "Введіть ім'я людини" << endl;
        cin >> Birthday.bName;  //Введення даних в поля структури
        cout << "Введіть число дня народження" << endl;
        cin >> Birthday.bDay;
        cout << "Введіть номер місяця народження" << endl;
        cin >> Birthday.bMonth;
        cout << "Введіть рік народження" << endl;
        cin >> Birthday.bYear;
        string &name = Birthday.bName;
        unsigned &day = Birthday.bDay;
        unsigned &month = Birthday.bMonth;
        unsigned &year = Birthday.bYear;
        cout << name << " за гороскопом " << fzodiac(month, day) << endl; //Виклик функції fzodiac з передачею їй параметрів посилань
        return 0;
}
```

### *Приклад створення зовнішньої функції.*

```
//zodiacFunc.cpp                Файл з кодом зовнішньої функції

#include "stdafx.h"
#include <string>
#include "Zodiac.h"

std::string fzodiac(const unsigned &fMonth, const unsigned &fDay) //Визначення власної функції з параметрами посиланнями на константи типу unsigned
{
        std::string zodiac;
        switch (fMonth) //Використання оператора switch для визначення знаку зодіаку в залежності від місяця та дня народження
        {
        case 1:
                fDay <= 20 ? zodiac = "Козеріг" : zodiac = "Водолій";
                break;
        case 2:
                fDay <= 18 ? zodiac = "Водолій" : zodiac = "Риби";
                break;
        case 3:
                fDay <= 20 ? zodiac = "Риби" : zodiac = "Овен";
                break;
        case 4:
                fDay <= 20 ? zodiac = "Овен" : zodiac = "Телець";
                break;
        case 5:
                fDay <= 21 ? zodiac = "Телець" : zodiac = "Близнюки";
                break;
        case 6:
                fDay <= 21 ? zodiac = "Близнюки" : zodiac = "Рак";
                break;
        case 7:
                fDay <= 22 ? zodiac = "Рак" : zodiac = "Лев";
                break;
        case 8:
                fDay <= 23 ? zodiac = "Лев" : zodiac = "Діва";
                break;
        case 9:
                fDay <= 23 ? zodiac = "Діва" : zodiac = "Терези";
                break;
        case 10:
                fDay <= 23 ? zodiac = "Терези" : zodiac = "Скорпіон";
                break;
        case 11:
                fDay <= 22 ? zodiac = "Скорпіон" : zodiac = "Стрілець";
                break;
        case 12:
                fDay <= 21 ? zodiac = "Стрілець" : zodiac = "Козеріг";
                break;
        };
        return zodiac; //Значення, яке повертає функція у місце свого виклику
}
```

***Приклад створення власної структури даних.***

```
// BirthdayStruct.h            Файл заголовку з об’явленням власної структури даних

#pragma once

#ifndef BIRTHDAYSTRUCT_H
#define BIRTHDAYSTRUCT_H
#include <string>
struct BirthdayStruct //Створення власної структури даних (класу)
{
        std::string bName;
        unsigned bDay;
        unsigned bMonth;
        unsigned bYear;
        float bAge;
        std::string bZodiac;
};
#endif
```

***Приклад об'явлення зовнішньої функції у файлу заголовку.***

```
//Zodiac.h      Файл заголовку з об'явленням зовнішньої функції

#pragma once
#ifndef ZODIAC_H
#define ZODIAC_H //Об'явлення змінної препроцесору
#include <string>
std::string fzodiac(const unsigned &, const unsigned &); //Об'явлення власної функції з параметрами
посиланнями на константи типу unsigned
#endif
```

## Контрольні питання

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.

***Загальні вимоги.***

1) Створити консольний додаток з ім'ям виду *PrizvischeKP5*.
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.***

***Частина 1.***

1) Запрограмувати роботу з рядковим типом *string*.
2) Звернення до елементів рядку організувати з використанням *ітераторів*.
3) Введення вхідних даних організувати в режимі діалогу з користувачем.
4) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП5_з1_OUT*.
5) Виведення даних у вихідних файл організувати в режимі додавання в кінець файлу.

***Частина 2.***

1) Запрограмувати роботу з одномірним *динамічним масивом* за допомогою вбудованого типу *масив*.
2) Звернення до елементів *масиву* організувати з використанням *покажчиків*.
3) Введення вхідних даних організувати в режимі діалогу з користувачем.
4) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП5_з2_OUT*.

5) Виведення даних у вихідних файл організувати в режимі додавання в кінець файлу.

## Теоретичні відомості

*Рядок* (string) – це послідовність символів змінної довжини. Для використання типу string, необхідно включити до коду заголовок string. Оскільки тип string належить бібліотеці, він визначений у просторі імен std.

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

Кожен клас визначає, як можуть бути ініціалізовані об'єкти його типу. Перелік найбільш розповсюджених ініціалізаторів об'єктів типу string наведений в табл. 5.1.

Таблиця 5.1 – Способи ініціалізації об'єкту класу string

| Ініціалізація | Роз'яснення |
|---|---|
| **string s1** | Ініціалізація за замовчуванням; s1 – пустий рядок |
| **string s2(s1)** | s2 - копія s1 |
| **string s2 = s1** | Еквівалент s2(s1), s2 - копія s1 |
| **string s3("value")** | s3 копія рядкового літерала, нульовий символ не включений |
| **string s3 = "value"** | Еквівалент s3("value"), s3 копія рядкового літерала |
| **string s4(n, 'c')** | Ініціалізація змінної s4 символом c в кількості n штук |

Коли змінна ініціалізується з використанням знаку =, компілятор копіює ініціалізуючий об'єкт в створюваний об'єкт, тобто відбувається *ініціалізація копією* (copy initialization). В іншому випадку, без знаку = та з використанням круглих дужок відбувається *пряма ініціалізація* (direct initialization).

Окрім способів створення та ініціалізації об'єктів клас визначає також операції, які можна виконувати з об'єктами класу. Найбільш розповсюджені операції класу string наведені в табл. 5.2.

Таблиця 5.2 – Операції класу string

| Операція | Роз'яснення |
|---|---|
| **os << s** | Виводить рядок s у потік виведення os. Повертає потік os |
| **is >> s** | Читає розділений пробілами рядок s з потоку is. Повертає потік is |
| **getline(is, s)** | Читає рядок введення з потоку is у змінну s. Повертає потік is |
| **s.empty()** | Повертає значення true, якщо рядок s пустий. Інакше повертає значення false |
| **s.size()** | Повертає довжину рядка, тобто кількість символів в ньому |
| **S[n]** | Повертає посилання на символ у позиції n рядку s; позиції відраховуються від 0 |
| **s1 + s2** | Повертає рядок, що складається з вмісту рядків s1 та s2 |
| **s1 = s2** | Замінює символи рядку s1 копією вмісту рядку s2 |
| **s1 == s2** | Рядки s1 та s2 рівні, якщо містять однакові символи. Регістр символів враховується |
| **s1 != s2** | Рядки s1 та s2 не рівні, якщо хоча б один символ чи регістр символу не співпадає |

| Операція | Роз'яснення |
|---|---|
| <, <=, >, >= | Порівняння залежить від регістру і покладається на алфавітний порядок символів |

За допомогою оператору циклу while можна організувати алгоритм зчитування невизначеної кількості рядків.

```
#include <iostream>
#include <string>
using std::string;
using std::cout;
using std::cin;
using std::endl;
int main()
{
	string word;
	while (cin >> word) //Читати до кінця файлу
	{
		cout << word << endl; //Відобразити кожне слово з нового рядку
	}
	return 0;
}
```

Умова оператора while перевіряє потік після завершення читання. Якщо потік припустимий, тобто не зустрівся символ кінця файлу (його можна задати сполученням клавіш ctrl + z під час введення) або неприпустиме значення, виконується тіло циклу while. В даному випадку воно виводить прочитане значення на стандартний пристрій виведення. Як тільки зустрічається кінець файлу (або неприпустиме введення), цикл while завершується. Аналогічно можна організувати введення невизначеної кількості даних іншого типу, наприклад int чи float.

Коли не потрібно ігнорувати пробіли у потоці введення, замість оператора >> слід використовувати функцію getline(). Функція getline() отримує потік введення і рядок. Функція читає наданий потік до першого символу нового рядку і зберігає прочитане без символу нового рядку в своєму аргументі типу string. Зустрівши символ нового рядку, навіть якщо він перший, функція getline() зупиняє читання і завершує роботу.

Подібно оператору введення, функція getline() повертає свій аргумент типу istream. В результаті функцію getline() можна використовувати в умові, як і оператор введення. Щоб програма з попереднього прикладу замість виведення по одному слову окремо вивела весь рядок, потрібно скористатись функцією getline().

```
int main()
{
	string line;
	while (getline(cin, line)) //Читати до кінця файлу
	{
```

```
            cout << line << endl; //Вивести весь введений рядок
        }
        return 0;
}
```

У класі `string` та деяких інших бібліотечних типах визначені допоміжні типи даних. Ці допоміжні типи дозволяють використовувати бібліотечні типи машинно-незалежним способом. Тип `size_type` –один з таких допоміжних типів. Для використання типу `size_type`, який визначений у класі `string`, застосовується *оператор області видимості* (оператор `::`), який вказує на те, що ім'я `size_type` визначене у класі `string`. Це *беззнаковий цілочисельний тип*, достатньо великий, щоб містити розмір любого рядку. Будь-яка змінна, котра використовується для збереження результату функції `size()` класу `string`, повинна мати тип `string::size_type`.

Для можливості виконання певних операцій з окремими символами рядка існує набір бібліотечних функцій, які визначені у заголовку `cctype`.

Таблиця 5.3 – Функції `cctype` для об'єктів типу `string`

| Функція | Операція |
|---|---|
| **isalnum(c)** | Повертає значення true, якщо с є літерою чи цифрою |
| **isalpha(c)** | Повертає значення true, якщо с є літерою |
| **iscntrl(c)** | Повертає значення true, якщо с – управляючий символ |
| **isdigit(c)** | Повертає значення true, якщо с є цифрою |
| **isgraph(c)** | Повертає значення true, якщо с не пробіл, а друкований символ |
| **islower(c)** | Повертає значення true, якщо с – символ в нижньому регістрі |
| **isprint(c)** | Повертає значення true, якщо с – друкований символ |
| **ispunct(c)** | Повертає значення true, якщо с – знак пунктуації |
| **isspace(c)** | Повертає значення true, якщо с – символ відступу (пробіл, табуляція, новий рядок і т.п.) |
| **isupper(c)** | Повертає значення true, якщо с – символ у верхньому регістрі |
| **isxdigit(c)** | Повертає значення true, якщо с є шістнадцятиричною цифрою |
| **tolower(c)** | Якщо с прописна літера – переводить її в нижній регістр, інакше залишає без змін |
| **toupper(c)** | Якщо с рядкова літера – переводить її у верхній регістр, інакше залишає без змін |

Існує два способи доступу до окремих символів в рядку: можна використовувати індексування чи ітератор.

*Оператор індексування* (оператор `[]`) отримує значення типу `string::size_type`, яке позначає позицію символу, до якого потрібен доступ. Оператор повертає посилання на символ у вказаній позиції.

Індексування рядків починається з нуля, першим символом буде `s[0]`, другим `s[1]`, а останнім `s[s.size() -1]`.

Значення оператора індексування називається індексом (index). Індекс може бути любим виразом, який повертає цілочисельне значення.

Для доступу до символів рядка також можна використовувати *ітератори* (`iterator`). Окрім рядків і векторів у всіх бібліотечних контейнерів є ітератори,

але тільки деякі з них підтримують оператор індексування. Тому ітератори є більш загальним та універсальним інструментом.

Як і вказівники, ітератори забезпечують опосередкований доступ до об'єкту. У випадку ітератора цим об'єктом є елемент в контейнері чи символ у рядку. Ітератор дозволяє вибрати елемент, а також підтримує операції переміщення з одного елементу на інший. Подібно покажчикам, ітератор може бути припустимим та неприпустимим. Припустимий ітератор вказує або на елемент, або на позицію за останнім елементом в контейнері. Всі інші значення ітератора неприпустимі.

На відміну від покажчиків, для отримання ітератора не потрібно використовувати оператор звернення до адреси (&). Для цього у типів, які мають ітератори, є члени, котрі повертають ці ітератори. А саме, вони мають функції-члени `begin()` та `end()`. Функція-член `begin()` повертає ітератор, який позначає перший елемент (або перший символ), якщо він є. Функція-член `end()` повертає ітератор, який вказує на наступну позицію за кінцем контейнеру (або рядку). Цей ітератор позначає неіснуючий елемент за кінцем контейнеру. Він використовується як індикатор того, що оброблені всі елементи контейнеру. Ітератор, який повертає функція `end()`, називають *ітератором після кінця* (`off-the-end iterator`), або скорочено *ітератором end*. Якщо контейнер пустий, функція `begin()` повертає той самий ітератор, що і функція `end()`.

Тип значень ітератора визначається в контейнері, для якого використовується ітератор. Наприклад, тип ітератора для рядкового типу буде `string::iterator`. Для визначення типу ітератора, як і багатьох інших типів, коли запис типу досить складний або невідомий до отримання значення виразу, зручно використовувати специфікатор `auto`.

```
string s;
//a позначає перший символ рядку s, b – елемент після останнього символу
auto a = s.begin(), b = s.end();
```

Ітератори підтримують лише кілька операцій, які перераховані в табл. 5.4.

Таблиця 5.4 – Стандартні операції з ітераторами контейнеру

| Операція | Пояснення |
|---|---|
| `*iter` | Повертає посилання на елемент, позначений ітератором `iter` |
| `iter->mem` | Звернення до значення ітератора `iter` і обробка члена `mem` основного елементу. Еквівалентне `(*iter).mem` |
| `++iter` | Інкремент ітератора `iter` для звернення до наступного елементу контейнеру |
| `--iter` | Декремент ітератора `iter` для звернення до попереднього елементу контейнеру |
| `iter1 == iter2` | Порівняння двох ітераторів. Два ітератори рівні, якщо вказують на один і той самий елемент контейнеру або обидва вказують на елемент після останнього того самого контейнеру |
| `iter1 != iter2` | Перевірка на нерівність двох ітераторів. Два ітератори нерівні, якщо вказують на різні елементи контейнеру або елементи різних контейнерів |

Подібно покажчикам, до значення ітератора можна звернутися (за допомогою оператора звернення до значення *), щоб отримати елемент, на який він посилається. Можна звертатись до значень лише припустимого ітератора.

Наступний код перетворює рядкові символи в прописні з використанням ітератора

```
string s("some string"); //Пряма ініціалізація рядкової змінної
//Заголовок оператора for, де задається початкове та кінцеве значення
//змінної циклу (ітератора) та її прирощення
for (string::iterator it = s.begin(); it != s.end(); it++)
{
	//Звернення до значення ітератора (символу рядка) та переведення
	//його у верхній регістр
	*it = toupper(*it);
}
cout << s << endl; //В результаті рядок матиме вигляд SOME STRING
```

В умові оператора `for` записаний логічний оператор `!=` замість оператора `<`. Це пов'язане з тим, що всі контейнери в мові C++ мають ітератори і для них визначені оператори `==` та `!=`. Але лише типи `string` та `vector` мають оператори індексування для яких також визначені оператори `<` та `>`. Щоб не думати про те, для якого контейнеру можна використовувати оператор `<`, програмісти C++ завжди обирають оператор `!=`.

Ітератори використовують оператор інкременту (оператор `++`) для переміщення з одного елементу на наступний. Операція прирощення логічно подібна прирощенню цілого числа. У випадку цілих чисел результатом буде цілочисельне значення збільшене на 1. У випадку ітераторів результатом буде переміщення ітератора на одну позицію вперед.

Ітератори рядків та векторів підтримують додаткові операції, що дозволяють переміщувати ітератори на декілька позицій за раз (наприклад, `iter +n`, `iter - n`, iter += n, `iter1 - iter2`). Вони також підтримують всі оператори порівняння. Ці оператори часто називають *арифметичними діями з ітераторами* (`iterator arithmetic`).

*Масив* (`array`) – це структура даних, яка дозволяє зберігати задану кількість *однотипних безіменних* елементів, до яких можна звертатись за позицією в масиві (індексом елементу). Масиви мають фіксований розмір і додавати елементи до масиву не можна.

Якщо завчасно не відома кількість елементів масиву, рекомендується використовувати вектор.

Масив є складеним типом. *Оператор об'явлення масиву* має форму `a[n]`, де a – ім'я масиву, n – розмірність масиву. Розмірність задає кількість елементів масиву, вона має бути більше нуля. Кількість елементів – це частина типу масиву,

тому вона має бути відома на момент компіляції (до запуску програми на виконання). Відповідно, розмірність має бути *константним виразом*.

```
int n = 10;
int a[n]; //Помилка!!! Розмірність має бути константним виразом
string a[10]; //Визначення масиву з 10 пустих рядків
```

Елементи масиву ініціалізуються значеннями за замовчуванням, якщо відсутня явна ініціалізація. Подібно змінним вбудованого типу, ініціалізований за замовчуванням масив вбудованого типу, який визначений у функції, буде містити невизначені значення.

При визначенні масиву необхідно вказати тип його елементів. Подібно вектору, масив містить об'єкти, тому неможливий масив посилань.

Масив підтримує ініціалізацію переліком значень. В цьому випадку розмірність можна опустити.

```
int a2[] = { 1,2,3 }; //Масив з трьох елементів цілочисельного типу
```

Не можна ініціалізувати масив копією іншого масиву, не можна також присвоїти один масив іншому.

Подібно бібліотечним типам `vector` та `string`, для доступу до елементів масиву можна використати *оператор індексування []* (`subscript`). Як завжди, індекси починаються з `0`. Для масиву з `10` елементів використовують індекси від `0` до 9.

При використанні змінної для індексування масиву її визначають типу `size_t`. Тип `size_t` – це машинно-незалежний беззнаковий цілочисельний тип, гарантовано великий для збереження розміру любого об'єкту в пам'яті. Тип `size_t` визначений у заголовку `cstddef`.

Якщо у виразі використати ім'я масиву без вказування індексу його елементу, компілятор замінить його покажчиком на перший елемент масиву. Покажчика на елементи масиву підтримують ті ж операції, що і ітератори. Наприклад можна використати оператор інкременту для переміщення на наступний елемент масиву

```
int arr[] = { 1,2,3,4,5 }; //Масив з п'яти елементів цілочисельного типу
int *p = arr; //Об'явлення покажчика на тип int та ініціалізація його
                // першим елементом масиву
++p; //Тепер покажчик вказує на другий елемент масиву
```

Покажчики подібно ітераторам можна використовувати для перебору елементів масиву. Для цього потрібно отримати покажчики на перший елемент масиву та елемент, наступний за останнім. Для масивів введені дві функції: `begin()` та `end()`. Ці функції працюють аналогічно однойменним функціям для контейнерів і дозволяють отримати покажчики на перший елемент масиву та наступний за останнім. Однак масиви не є класами, тому не можуть містити функцій-членів. Через це для роботи ці функції в якості аргументу

використовують масив (наприклад, `begin(arr)`). Ці функції визначені в заголовці `iterator`.

В мові C++ *багатомірних масивів* (`multidimensioned array`) не існує. Те, що згадують як багатомірний масив в дійсності є *масивом масивів*.

При визначенні масиву, елементи якого є масивами, вказуються дві розмірності: розмірність самого масиву та розмірність його елементів.

```
//Масив з 3 елементів, кожен з яких є масивом з 4 цілих чисел
int arr1[3][4];
//Масив з 10 елементів, кожен з яких є масивом з 20 елементів,
//кожен з яких є масивом з 30 цілих чисел ініціалізованих 0
int arr2[10][20][30] = {0};
```

Подібно любим масивам, елементи багатомірного масиву можна ініціалізувати, навівши у фігурних дужках перелік ініціалізаторів. Багатомірні масиви можуть бути ініціалізованіпереліками значень у фігурних дужках для кожного рядку.

```
//Масив з 3 елементів, кожен з яких є масивом з 4 цілих чисел
int arr1[3][4] =
{
      {0, 1, 2, 3},    //ініціалізатори рядку 0
      {4, 5, 6, 7},    //ініціалізатори рядку 1
      {8, 9, 10, 11}   //ініціалізатори рядку 2
};
```

Для доступу до багатомірного масиву можна використовувати індексацію.

```
//Розмірність масиву має бути константним виразом
constexpr size_t rowCnt = 3, colCnt = 4;
//Двомірний масив з 12 неініціалізованих елементів
int arr[rowCnt][colCnt];
for (size_t i = 0; i != rowCnt; i++) // Для кожного рядку
{
      // Для кожного стовпчику в рядку
      for (size_t j = 0; j != colCnt; j++)
      {
            //Присвоїти елементу його індекс як значення
            arr[i][j] = i * colCnt + j;
      }
}
```

У випадку коли розмірність масиву до запуску програми невідома, можна скористатись *динамічним масивом*. Динамічний масив можна задати з використанням покажчиків за допомогою наступної програмної конструкції

```
size_t n; //Об’явлення змінної для розмірності масиву
cout << "Задайте розмірність масиву" << endl;
cin >> n; //Запис розмірності масиву у змінну n
int *arr =new int [n]; //Створення масиву arr для n цілих чисел
for (int *p = arr; p != arr+n; p++) //Запис значень в масив
{
```

```
        cout << "Задайте елемент масиву" << endl;
        cin >> *p; //Введення значення елементу масиву
}
delete[] arr; //Звільнення динамічної пам'яті, яка була виділена під масив
```

При створенні динамічного масиву використовують *оператор* new[] – це оператор створення масиву. Він має синтаксис

```
new тип [розмірність]
```

Для звернення до елементів масиву створюється змінна типу покажчик на масив з елементами певного типу

```
int *arr =new int [n];
```

Обов'язковою умовою використання динамічних масивів є *звільнення пам'яті*, яка була виділена для збереження елементів масиву, після завершення роботи з масивом. Ця операція виконується за допомогою *оператора* delete з пустими квадратними дужками та іменем динамічного масиву без вказування індексу чи кількості елементів:

```
delete[] arr;
```

Якщо не застосувати оператор delete у зазначеному форматі, масив буде займати оперативну пам'ять комп'ютера навіть після завершення роботи програми. При зазначенні певного індексу у квадратних дужках після імені масиву в операторі delete звільниться лише пам'ять, яку займав елемент масиву з вказаним індексом. Оператор delete без квадратних дужок використовується для звільнення пам'яті, що була виділена для вказаного після оператора delete об'єкту.

## Приклад програмної реалізації

***Приклад роботи з рядковим типом string та застосуванням ітераторів (Частина 1).***

```
// Запам’ятовування окремих слів без пробілів з введеного речення у вектор
// і виведення елементів вектору по 8 слів у рядку


#include "stdafx.h"
#include <iostream>
#include <string>
#include <vector>
#include "windows.h"            //Необхідно для використання функцій SetConsoleCP() та SetConsoleOutputCP()
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;

int main() //використання ітератора для виведення текстових елементів вектора
{
        SetConsoleCP(1251);                     //Необхідно для введення в консолі української мови
        SetConsoleOutputCP(1251);               //Необхідно для виведення в консолі української мови
        string word, line, text;
        vector<string> vect1;
        while (getline(cin, line)) //Зчитуємо текст рядками в змінну text поки не буде введено ctrl+z
        {
                text += line + " ";
        }
        cout << "Текст зчитано" << endl;
        cout << text << endl;
        for (string::iterator i = text.begin(); i != text.end(); i++) //Розбиваємо текст на слова
        {
                if (isgraph(*i)) //Запам'ятовуємо окреме слово в змінну word
                {
                        word += *i;
                }
                else
                {
                        if (word.size() != 0) //Якщо слово закінчилось, додаємо його окремим елементов у
                                              //вектор vect1
                        {
                                vect1.push_back(word);
                                word = "";
                        }
                }
        }
        cout << endl;
        cout << "Виводимо елементи вектору по 8 слів у рядку" << endl;
        unsigned num8 = 0;
        //Виводимо елементи вектора на екран по 8 в рядку
        //за допомогою ітератора замість індексації
        for (vector<string>::const_iterator i = vect1.begin(); i != vect1.end(); i++)
        {
                cout << *i << " ";
                num8++;
                if (num8 == 8)
                {
                        cout << endl;
                        num8 = 0;
                }
        }
```

```
        cout << endl;
        return 0;
}
```

***Приклад роботи з одномірним динамічним масивом за допомогою вбудованого типу масив та застосуванням покажчиків (Частина 2).***

```
// Створення динамічного масиву та запис в нього значень
//ініціалізація вектору масивом

#include "stdafx.h"
#include <iostream>
#include <vector>
#include "windows.h"            //Необхідно для використання функцій SetConsoleCP() та SetConsoleOutputCP()
using std::cin;
using std::cout;
using std::endl;
using std::vector;


int main() //Робота з динамічним масивом дійсних чисел та копіювання його значень у вектор
{
        SetConsoleCP(1251);                     //Необхідно для введення в консолі української мови
        SetConsoleOutputCP(1251);               //Необхідно для виведення в консолі української мови
        size_t n, ii=0;
        cout << "Задайте розмірність масиву" << endl;
        cin >> n;
        double *arr =new double [n];
        for (double *arP = arr; arP < arr+n; arP++) //Запис значень в масив
        {
                cout << "Задайте елемент масиву" << endl;
                cin >> *arP;
        }
        cout << endl;
        cout << "Елементи масиву" << endl;
        for (double *arP = arr; arP < arr + n; arP++) //Виведення елементів масиву на екран
        {
                cout << "arr[" << ii << "]=" << *arP << endl;
                ii++;
        }
        cout << endl;
        vector<double> vec(arr, arr+n); //Ініціалізація вектора масивом
        cout << "Елементи вектора" << endl;
        ii = 0;
        for (vector<double>::iterator iter = vec.begin(); iter != vec.end(); iter++)
        {
                cout << "vec[" << ii << "]=" << *iter << endl;
                ii++;
        }
        delete[] arr; //Звільнення динамічної пам'яті, яка була виділена під масив

        return 0;
}
```

## Контрольні питання

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.

***Загальні вимоги.***

1) Створити консольний додаток з ім'ям виду *PrizvischeKP6*.
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.***

***Частина 1.***

1) Запрограмувати роботу з одномірним *масивом* за допомогою типу *vector*.
2) Звернення до елементів *вектору* організувати з використанням *оператора індексації*.
3) Введення вхідних даних організувати в режимі діалогу з користувачем.
4) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП6_з1_OUT*.
5) Виведення даних у вихідних файл організувати в режимі додавання в кінець файлу.

***Частина 2.***

1) Запрограмувати роботу з двомірним *динамічним масивом* за допомогою вбудованого типу *vector*.
2) Звернення до елементів *вектору* організувати з використанням *ітераторів*.
3) Введення вхідних даних організувати з вхідного файлу виду *ПрізвищеКП6_з2_INP*.

4) Завдання для обчислень, вхідні дані та результати розрахунків вивести у зовнішній текстовий файл з ім'ям виду *ПрізвищеКП6_з2_OUT*.
5) 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 6.1 – 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 |

| Ініціалізація | Роз'яснення |
|---|---|
| `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 замовчувнненням, 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 6.2 – 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.

## Приклад програмної реалізації

***Приклад роботи з одномірним масивом за допомогою типу vector та застосуванням індексації (Частина 1).***

```
// Розрахувати суми 1-го і останнього, 2-го і передостаннього … елементів вектору

#include "stdafx.h"
#include <iostream>;
#include <vector>;
#include "windows.h"            //Необхідно для використання функцій SetConsoleCP() та SetConsoleOutputCP()
using std::cin;
using std::cout;
using std::endl;
using std::vector;

int main()
{
        SetConsoleCP(1251);                     //Необхідно для введення в консолі української мови
        SetConsoleOutputCP(1251);               //Необхідно для виведення в консолі української мови
        vector<int> intV;
        int num;
        while (cin >> num) //Задати вектор з довільних цілих чисел
        {
                intV.push_back(num);
        }

        for (vector<int>::size_type i = 0; i != intV.size(); i++) //Просумувати 1-й і останній, 2-й та
                                                                  //передостанній і т.д. елементи вектору
        {
                if (i<(intV.size() - i - 1)) //шукаємо середину вектора
                {
                        cout << "Сума двох протилежних елементів вектору " << intV[i] << " та "
                                << intV[intV.size() - i - 1]
                                << " = " << intV[i] + intV[intV.size() - i - 1] << endl;
                }

        }
        return 0;
}
```

***Приклад роботи з двомірним динамічним масивом за допомогою типу vector та застосуванням ітераторів (Частина 2).***

```
// Створення двомірного вектору та запис в нього значень
//виведення елементів двомірного вектору на екран

#include "stdafx.h"
#include <iostream>
#include <vector>
#include "windows.h"            //Необхідно для використання функцій SetConsoleCP() та SetConsoleOutputCP()
using std::cin;
using std::cout;
using std::endl;
using std::vector;


int main() //Ілюстрація використання двомірних векторів для задавання динамічних матриць (замість динамічних двомірних масивів)
{
        SetConsoleCP(1251);                     //Необхідно для введення в консолі української мови
        SetConsoleOutputCP(1251);               //Необхідно для виведення в консолі української мови
        size_t n, m;
        cout << "Задайте розмірність матриці n - рядки" << endl;
        cin >> n;
        cout << "Задайте розмірність матриці m - стовпчики" << endl;
        cin >> m;
        cout << endl;
        typedef vector<vector<double>> StrMat; //Створення нового типу, вектору що складається з векторів
                                               //дійсних чисел (зовнішній вектор)
        typedef vector<double> StrVec;  //Створення нового типу - вектору з дійсних чисел (внутрішній вектор)
        StrMat Matrix;                  //Створення нової змінної типу зовнішнього вектору
        StrVec Vect;                    //Створення нової змінної типу внутрішнього вектору
        for (size_t i = 0; i != n; i++) //Задавання елементів зовнішнього вектору
                {
                        for (size_t j = 0; j != m; j++) //Задавання елементів внутрішнього вектору
                        {
                                //Залежність для розрахунку елементу внутрішнього вектору
                                //і додавання його в кінець
                                Vect.push_back((cos(j * m + i * n)+ m) / (sin(j * n + i * m) + n));
                        }
                        Matrix.push_back(Vect); //Додавання в зовнішній вектор елементу
                                                //(внутрішній вектор) в кінець
                        Vect.clear(); //Очищення внутрішнього вектору для додавання нових елементів
                }
        cout << "Результуюча матриця" << endl;
        //Використання ітератора для зовнішнього вектору
        for (StrMat::iterator MatIter = Matrix.begin(); MatIter != Matrix.end(); MatIter++)
        {
                //Використання ітератора для внутрішнього вектору
                for (StrVec::iterator VecIter = MatIter->begin(); VecIter != MatIter->end(); VecIter++)
                {
                        cout << *VecIter << " "; //Виведення на екран елементів вектору в рядок
                }
                cout << endl; //Перехід на новий рядок для виведення наступного рядку матриці
        }
        cout << endl;
        return 0;
}
```

## 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?

# 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.

***Загальні вимоги.***

1) Створити консольний додаток з ім'ям виду *PrizvischeKP7*.
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) Передбачити *обробку виключень* в програмі за допомогою блоків *try* та *catch*.
2) Включити в програму кілька блоків *catch* для обробки різних *типів виключень*.
3) Використати оператор *throw* для передачі *виключень* у блоки *catch*.
4) Передбачити введення вхідних та виведення вихідних даних за допомогою зовнішніх файлів.
5) Перед введенням вхідних даних запропонувати користувачу вибір джерела введення – з зовнішнього файлу чи з клавіатури.
6) За допомогою оператора *switch* організувати перебір потрібних варіантів згідно свого завдання.
7) Застосувати *оператори переходу break* та (або) *continue* для переривання роботи оператора *switch* та *операторів циклу*.
8) Для збереження результатів розрахунку використати тип *vector* та власну *структуру даних*, створену у *файлі заголовку*.
9) Результати розрахунку вивести на екран та (або), на запит користувача, у зовнішній файл в режимі додавання в кінець.
10) За можливості компілятора використати в програмі *серійний оператор циклу for*.

11) По завершенню розрахунків запропонувати користувачу з застосуванням оператору циклу *do while* вибір: завершити роботу програми чи запустити її на повторний розрахунок.

## Теоретичні відомості

*Оператори* (statement) – це прграмні елементи, які контролюють як і в якій послідовності виконуватимуться вирази. В мові C++ визначений набір операторів *керування потоком* (flow of control), які забезпечують більш складні шляхи виконання коду ніж послідовне.

Більшість операторів в мові C++ закінчуються крапкою з комою. Вираз типу ival + 5 стає *оператором виразу* (expression statement), який завершується крапкою з комою. Як правило, вирази містять оператори, результат обчислень яких впливає на стан програми. До таких операторів відносяться присвоєння, інкремент, введення та виведення.

Сама проста форма оператора – це *пустий* (empty), або *нульовий оператор* (null statement). Він представляє собою символ крапки з комою (;). Пустий оператор використовують у випадку, коли синтаксис мови потребує наявність оператора, а логіка програми – ні. Зазвичай це відбувається у випадку, коли вся робота циклу здійснюється в його умові. Наприклад, можна організувати введення, ігноруючи всі прочитані дані, поки не зустрінеться певне значення:

```
//цикл виконується доки не зустрінеться кінець файлу
//або значення рівне змісту змінної sought
while (cin >> s && s != sought)
      ; //пустий оператор
```

В більшості випадків пустий оператор безпечний, але помилкова крапка з комою в кінці умови циклу while або оператора if може призвести до помилкового коду:

```
vector<string> svec;
vector<string>::iterator iter;
while (iter != svec.end()); //тіло циклу пусте!
      ++iter; // оператор інкременту не є частиною циклу
```

*Складений оператор* (compound statement), зазвичай називають *блоком* (block), він являє послідовність операторів, які заключені у *фігурні дужки*. Об'єднавши кілька операторів, узявши їх у фігурні дужки, можна отримати блок операторів, який розглядається як один оператор. Блок операторів має власну *область видимості*. Об'явлені у блоці імена доступні тільки у даному блоці і блоках, вкладених у нього. Як зазвичай, ім'я видиме тільки з моменту коли воно визначене і до кінця блоку включно. В кінці блоку не ставиться крапка з комою.

Змінні можна визначати в керуючих структурах операторів if, switch, while та for. Змінні, які визначені у керуючій структурі, видимі лише в межах

цього оператора. Якщо до змінної потрібно звертатися після роботи керуючого оператора, то її слід визначити поза межами оператора.

Мова C++ включає два оператори, які забезпечують умовне виконання. *Оператор* `if` розділяє потік виконання на основі умови. *Оператор* `switch` обраховує результат цілочисельного виразу і на його основі обирає один з кількох шляхів виконання.

Одним з різновидів умовного оператора є *умовний оператор* (`?:`). Він аналогічний оператору `if else`. Синтаксис оператора наступний:

```
умова ? вираз1 : вираз2;
```

Умовний оператор `?:` є тернарним оператором (тобто має три операнди). Умовний оператор працює наступним чином:

- перший операнд (умова) обраховується і неявно перетворюється в `bool`;
- якщо результатом першого операнду буде значення `true` (1), обчислюється другий операнд (вираз1);
- якщо результатом першого операнду буде значення `false` (0), обчислюється третій операнд (вираз2).

Умовний оператор `?:` може застосовуватись там, де використання оператора `if else` неможливе, наприклад в операторі return якоїсь функції

```
return ival == 10 ? 1 : 0; //if else використовувати тут не можна
```

або в операторі виводу

```
cout << (ival < 100 ? "ival<100" : "ival>=100"); //з if таке не можливе
```

Оператор `switch` надає зручний спосіб вибору однієї з кількох альтернатив. *Оператор* `switch` – оператор умовного виконання, котрий спочатку обраховує результат виразу, що йде за ключовим словом `switch`, а потім передає управління розділу `case`, мітка якого співпадає з результатом виразу. Коли відповідної мітки немає, виконання переходить до розділу `default` (якщо він є) або до наступного оператора, який йде за оператором `switch`, при відсутності розділу `default`. Оператор `switch` має наступний синтаксис

```
switch (вираз)
{
case мітка1:
      вираз1;
      break;
case мітка2:
      вираз2;
      break;
default:
      вираз3;
      break;
}
```

Результат виразу, який міститься в круглих дужках після ключового слова switch повинен мати *цілочисельний тип* (чи можливість неявного перетворення в цілочисельний тип) або *тип enum* (тип перерахування - перелік значень, які мають чітко визначену послідовність).

В кінці кожного блоку case та блоку default необхідно застосовувати оператор break для переривання роботи оператора switch. В противному випадку після співпадіння мітки і виконання відповідного блоку case, продовжиться виконання наступних блоків case. *Оператор break* перериває потік виконання і передає управління першому оператору після оператора switch.

Ключове слово case і пов'язане з ним значення називають також *міткою case* (case label). Значення кожної мітки case має бути *константним виразом*.

```
char ch;
int ival = 28;
switch (ch)
{
case 3.14: //Помилка! Мітка не ціле число
      //Вираз
      break;
case ival: //Помилка! Мітка не константа
      //Вираз
      break;
default:
      //Вираз
      break;
}
```

Однакові значення міток неприпустимі.

*Ітераційні оператори* (iterative statement), називають також *циклами* (loop). Вони забезпечують повторне виконання коду, доки їх умова істинна. Оператори while та for перевіряють умову до виконання тіла циклу (*цикли з передумовою*). Оператор do while спочатку виконує тіло циклу, а потім перевіряє свою умову (*цикл з післяумовою*).

*Серійний оператор for*, на відміну від традиційного оператора for, створений для спрощення перебирання елементів контейнеру чи іншої послідовності. Синтаксис серійного оператору for (range for) наступний:

```
for (об'явлення: вираз)
      оператор
```

*Об'явлення* визначає змінну. Кожен елемент послідовності повинен допускати перетворення в тип змінної. Простіше за все гарантувати відповідність типів за рахунок використання *специфікатору типу* auto. Так компілятор

виведе тип сам. Якщо необхідний запис в елементи послідовності, то змінна циклу повинна мати *тип посилання*.

*Вираз* має надати деяку послідовність, таку, як список ініціалізації, масив, або об'єкт такого типу, як `vector` чи `string`, у якого є функції-члени `begin()` та `end()`, котрі повертають ітератори.

На кожній ітерації керуюча змінна визначається та ініціалізується наступним значенням послідовності, а потім виконується *оператор*. Як зазвичай, оператор може бути *одиночним оператором* або *блоком*. Виконання завершується, коли всі елементи оброблені.

В наступному прикладі цикл подвоює значення кожного елементу вектору:

```
vector<int> v = { 0,1,2,3,4,5,6,7,8,9 };
//для запису в елементи змінна діапазону має бути посиланням
for (auto &r : v) //для кожного елементу вектора v
    r *= 2; //подвоїти значення кожного елементу
```

Заголовок `for` об'являє, що керуюча змінна циклу `r` пов'язана з вектором `v`. Щоб дозволити компілятору самостійно вивести тип змінної `r`, використовуємо специфікатор типу `auto`. Оскільки передбачається зміна значень елементів вектору `v`, об'являємо змінну `r` як посилання за допомогою оператора `&`. При присвоєнні їй значень в циклі фактично присвоюється значення елементу, з яким пов'язана змінна `r` на даний момент.

З використанням традиційного циклу for та без застосування специфікатора auto попередній цикл мав би наступний вигляд:

```
for (vector<int>::iterator r = v.begin(); r != v.end(); r++)
    *r *= 2;
```

В даному варіанті в заголовку циклу `for` визначається ітератор `r` типу `vector<int>::iterator`; задається початкове значення ітератора `r` за допомогою функції-члена `begin()`; перевіряється умова, чи не досяг ітератор кінцевого елементу вектора, за допомогою функції-члена `end()`; та задається прирощення ітератора `r` за допомогою оператору постфіксного інкременту `++`. В тілі циклу подвоюється значення поточного елементу вектора з використанням оператора звернення до значення `*` та складеного оператора присвоєння `*=`.

Серійний оператор `for` неможливо використовувати для додавання елементів у вектор чи інший контейнер. В серійному операторі `for` кешується (запам'ятовується) значення `end()`. Якщо додати чи видалити елементи з послідовності, збережене значення `end()` стане невірним.

*Оператор `do while`* схожий на оператор `while`, але його умова перевіряється після виконання тіла циклу. Незалежно від значення умови тіло циклу *виконується принаймні один раз*. Його синтаксична форма наведена нижче.

```
do
{
```

```
        оператор
} while (умова);
```

В циклі do while *оператор* виконується перше, ніж перевіриться *умова*. При цьому умова не може бути пустою. Якщо *умова* брехня, цикл завершується, в іншому разі цикл повторюється. Змінні, які використовуються в умові, слід визначити за межами циклу do while.

Наступний приклад сумує будь-яку кількість пари чисел за допомогою циклу do while.

```
string rsp;
do
{
        cout << "Введіть два значення:";
        int val1 = 0, val2 = 0;
        cin >> val1 >> val2; //значення вводяться через пробіл
        cout << "Сума " << val1 << " та " << val2 << " = " << val1 + val2
             << "\n" << "Продовжити введення? Введіть ‘так’ чи ‘ні’: ";
        cin >> rsp;
} while (!rsp.empty() && rsp[0] != 'н');
```

Цикл починається запитом у користувача двох чисел. Потім виводиться їх сума і іде запит, чи бажає користувач підсумувати далі. Відповідь користувача перевіряється в умові. Якщо введення пусте або починається з н, цикл завершується. В іншому випадку цикл повторюється.

Оскільки умова не обробляється до завершення оператору чи блоку, цикл do while *не дозволяє визначати змінні в умові*. Якщо визначити змінні в умові, то будь-яке їх використання відбудеться *до визначення*.

*Оператори переходу* перериває потік виконання. Мова C++ надає чотири оператори: break, continue, goto та return.

*Оператор break* завершує найближчий оточуючий оператор while, do while, for або switch. Виконання поновлюється з оператора, що знаходиться безпосередньо за оператором, робота якого завершується.

Оператор break може розташовуватись лише в циклі або операторі switch (включно з операторами або блоками, вкладеними в ці цикли). Оператор break впливає лише на найближчий оточуючий цикл або оператор switch.

*Оператор continue* перериває поточну ітерацію найближчого циклу і негайно починає наступну. Оператор continue може знаходитись лише в циклах while, do while або for, включно з операторами або блоками, вкладеними в ці цикли. Однак, на відміну від оператора break, оператор continue може бути присутнім в операторі switch лише як складова вбудованого ітераційного циклу.

Оператор continue перериває лише поточну ітерацію; виконання залишається в циклі. У випадку циклів while або do while виконання

продовжується з оцінки умови. В традиційному циклі `for` виконання продовжується у виразі заголовку. В серійному операторі `for` виконання продовжується з ініціалізації керуючої змінної наступним елементом послідовності.

Наступний цикл читає зі стандартного пристрою введення по одному слову за раз. Оброблені будуть тільки ті слова, які починаються з символу підкреслення. Для любого іншого значення поточна ітерація завершується.

```
string buf;
while (cin >> buf && !buf.empty())
{
	if (buf[0] != '_')
	{
		continue; //отримати наступне значення
	}
}
```

*Оператор `goto`* забезпечує безумовний перехід до іншого оператору в тій же функції. Оператор `goto` ускладнює розуміння та можливість зміни програми, тому його використання не рекомендоване.

*Виключення* (`exception`) – це анамалії часу виконання, такі як втрата підключення до бази даних або введення непередбачуваних даних, які порушують нормальне функціонування програми. Також від виключенням розуміють об’єкт системного класу чи класу користувача, який створений операційною системою чи кодом програми у відповідь на обставини, які не дозволяють подальше нормальне виконання програми. Обробка виключень в додатку дозволяє коректно вийти із скрутної ситуації.

Обробка виключень зазвичай використовується у випадку, коли деяка частина програми виявила проблему, з якою вона не може впоратись. Причому характер проблеми такий, що не дає змоги продовжити виконання частині програми, яка виявила цю проблему. В такому випадку ділянка програми, яка виявила проблему, потребує спосіб повідомити про те що сталося та про те, що ця ділянка програми не здатна продовжити виконання. Повідомивши про те що сталося, ділянка програми, яка виявила виключення, припиняє роботу.

Кожній частині програми, здатній передати виключення, відповідає інша частина, код якої здатний обробити виключення, незалежно від того, що сталося. Наприклад, якщо проблема у неприпустимому вводі, то частина обробки могла б попросити користувача ввести правильні дані.

Виключення забезпечують взаємодію частин програми, які виявили проблему та які вирішують її. Обробка виключень у мові C++ має на увазі наступне.

- *Оператор throw* використовується частиною коду, що виявила проблему, з якою вона не може впоратись. Про оператор throw говорять, що він *передає* (raise) виключення.
- *Блок try* використовується частиною обробки виключення. Блок try починається з *ключового слова try* і завершується однією чи декількома *директивами catch* (catch clause). Виключення, які передані з коду, що розміщений у блоці try, як правило, обробляються в одному з розділів catch. Оскільки розділи catch обробляють виключення, їх називають також *обробниками виключень* (exception handler).
- Набір визначених у бібліотеці *класів виключень* (exception class) використовується для передачі інформації про те що відбулося поміж операторами throw і відповідними розділами catch.

Частина програми, що виявила виключення, використовує оператор throw для передачі виключення. Він складається з ключового слова throw, що супроводжується виразом. Вираз визначає тип виключення, яке передається. Оператор throw, як правило, завершується крапкою з комою, що робить його виразом.

```
if (!infile)    //Якщо не вдалося відкрити вхідний файл -
                //ініціалізувати виключну ситуацію
{
     throw std::runtime_error("Помилка відкриття вхідного файлу.\n");
     return -1;
};
```

Тип runtime_error є одним з типів виключень, які визначені у заголовку stdexcept стандартної бібліотеки. Об'єкт класу runtime_error слід ініціалізувати об'єктом класу string. Цей рядок надає додаткову інформацію про проблему.

Блок try має наступний синтаксис:

```
try
{
     //оператори_програми
}
catch (об'явлення_виключення)
{
     //оператори_обробника
}
catch (об'явлення_виключення)
{
     //оператори_обробника
} //…
```

Блок try починаєтьс з ключового слова try, за яким іде блок коду у фігурних дужках, у якому потенційно можуть виникнути виключення.

Блок `try` супроводжується одним чи кількома блоками `catch`. Блок `catch` складається з трьох частин: ключового слова `catch`, об'явлення (можливо, безіменного) об'єкту у круглих дужках, що зветься *об'явленням виключення* (exception declaration), і операторного блоку. Коли об'явлення виключення у блоці `catch` співпадає з виключенням, виконується пов'язаний з ним блок. По завершенню виконання коду обробника управління переходить до наступного за ним оператора.

Оператори_програми у блоці `try` є звичайними програмними операторами, що реалізують логіку програми. Об'явлені в блоці `try` змінні недоступні поза блоком, в тому числі, недоступні і в блоках `catch`.

В бібліотеці C++ визначений набір класів, об'єкти яких можна використовувати для передачі повідомлень про проблеми у функціях, що визначені в стандартній бібліотеці. Ці стандартні класи виключень можуть бути токож використані у програмах, створюваних розробником. Бібліотечні класи виключень визначені у чотирьох заголовках.

- В заголовку `exception` визначений загальний клас виключень `exception`. Він повідомляє лише про те, що виключення виникло, але не надає ніякої додаткової інформації.
- В заголовку `stdexcept` визначено декілька універсальних класів виключень (табл. 7.1).
- В заголовку new визначений клас `bad_alloc`.
- В заголовку `type_info` визначений клас виключення `bad_cast`.

В класах `exception`, `bad_alloc` та `bad_cast` визначений лише стандартний конструктор, тому неможливо ініціалізувати об'єкти цих типів.

Поведінка виключень інших типів прямо протилежна: їх можна ініціалізувати об'єктом класу `string`, однак значенням за замовчуванням їх ініціалізувати *не можна*. При створенні об'єкту виключення любого з цих типів необхідно надати ініціалізатор. Цей ініціалізатор використовується для надання додаткової інформації про помилку.

Таблиця 7.1 – Стандартні класи виключень заголовку `stdexcept`

| Клас | Роз'яснення |
|---|---|
| **exception** | Найбільш загальний вид проблеми |
| **runtime_error** | Проблема, яка може бути виявлена лише під час виконання |
| **range_error** | Помилка часу виконання: отриманий результат перевищує припустимий діапазон значень |
| **overflow_error** | Помилка часу виконання: переповнення регістру при обчисленні |
| **underflow_error** | Помилка часу виконання: недоповнення регістру при обчисленні |
| **logic_error** | Помилка в логиці програми |
| **domain_error** | Логічна помилка: аргумент, для якого не існує результату |
| **invalid_argument** | Логічна помилка: непридатний аргумент |
| **length_error** | Логічна помилка: спроба створити об'єкт більшого розміру, ніж максимально припустимий для даного типу |

| Клас | Роз'яснення |
|---|---|
| out_off_range | Логічна помилка: значення поза межами допустимого діапазону |

У класах виключень визначена лише одна функція what(). Вона не отримує ніяких аргументів і повертає константний покажчик на тип char. Це покажчик на символьний рядок у стилі C, який містить текст опису переданого виключення.

```
try
{
	//оператори_програми
}
catch (const std::runtime_error err)		//Перехоплення виключень
{
	cout << "Виникла помилка в процесі виконання програми." << endl;
	cout << err.what() << endl;
}
catch (const std::out_of_range err)		//Перехоплення виключень
{
	cout << "Виникла помилка. Значення поза межами діапазону." << endl;
	cout << err.what() << endl;
}
```

У кожному додатку має бути передбачена обробка виключних ситуацій для забезпечення стабільної, безвідмовної роботи.

## Приклад програмної реалізації

***Приклад роботи з умовними операторами, операторами циклу та переходу, а також оператором throw та блоком try***

```
//Пошук та заміна літер у введених словах
//

#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <stdexcept>
#include "windows.h"            //Необхідно для виведення в консолі української мови
#include "strVec.h"             //Підключення файлу заголовку зі створеною структурою даних
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::vector;
using std::ofstream;
using std::ifstream;

int main()
{
        SetConsoleCP(1251);             //Необхідно для введення в консолі української мови
        SetConsoleOutputCP(1251);       //Необхідно для виведення в консолі української мови
        ofstream outfile("out_file.txt", ofstream::app);//Відкриття файлу для додавання в кінець
        ifstream infile("in_file.txt");
        char n;
        do
        {
                try
                {
                        string text, line, wrd, wrd123;
                        strVec sVec, sVec123;
                        vector<strVec> vect;
                        //Запит на запуск повторного розрахунку
                        cout << "Задати речення з вхідного файлу? [y] [n]" << endl;
                        //Якщо не вдалося відкрити вихідний файл - ініціалізувати виключну ситуацію
                        if (!outfile)
                        {
                                throw std::runtime_error("Помилка відкриття вихідного файлу.\n");
                                return -1;
                        };
                        cin >> n;
                        if (n=='y')
                        {
                                if (!infile)    //Якщо не вдалося відкрити вхідний файл -
                                                //ініціалізувати виключну ситуацію
                                {
                                        throw std::runtime_error("Помилка відкриття вхідного файлу.\n");
                                        return -1;
                                };
                                while (getline(infile, line))   //Зчитуємо текст рядками в змінну text
                                                                //поки не буде введено ctrl+z
                                {
                                        text += line + " ";     //Записуємо всі введені символьні рядки
                                                                //в змінну text
                                }
                        }
                        else
                        {
                                cout << "Задайте речення. Завершення вводу ctrl+z" << endl;

                                while (getline(cin, line))      //Зчитуємо текст рядками в змінну text
                                                                //поки не буде введено ctrl+z
                                {
                                        text += line + " ";     //Записуємо всі введені символьні рядки
                                                                //в змінну text
                                }
```

```
	}
	for (string::size_type i = 0; i != text.size(); i++) //Розбиваємо текст на слова
	{
		if (isgraph(text[i]))   //Перевіряємо чи поточний символ графічний
		{
			wrd += text[i]; //Додаємо поточну літеру в змінну wrd
			switch (text[i])//Шукаємо потрібні літери, рахуємо їх кількість
					//та замінюємо на цифри
			{
			case 'a':                       //якщо поточна літера a
				++sVec.count_a;         //лічільник літер a збільшеємо на 1
				text[i] = '1';          //заміна літери a на 1
				break;                  //переривання роботи оператора switch
			case 'o':
				++sVec.count_b;
				text[i] = '2';
				break;
			case 'e':
				++sVec.count_c;
				text[i] = '3';
				break;
			default:
				++sVec.count_n;
				break;
			}
			wrd123+= text[i];       //Додаємо поточну літеру з урахуванням заміни
						//в змінну wrd123
		}
		else if (wrd.size() != 0)       //Якщо слово закінчилось, додаємо його
						//окремим елементов у змінну sVec.word
		{
				sVec.word=wrd;  //Запам'ятовуємо слово без замін літер
						//у змінну sVec.word
				wrd = "";
				sVec.word123 = wrd123;  //Запам'ятовуємо слово після замін
							//літер у змінну sVec.word123
				wrd123 = "";
				vect.push_back(sVec);   //Додаємо у вектор змінну типу
				//створеної структури даних, яка містить вхідне та вихідне
				//слово та підраховану кількість літер
				sVec.count_a = 0;
				sVec.count_b = 0;
				sVec.count_c = 0;
				sVec.count_n = 0;
		}
		else if (text[i] < -1 || text[i] > 255) //Діапазон латинських літер
							//від -1 до 255
		{
			throw std::out_of_range("Речення треба вводити латинськими літерами.
\n");
			return -1;
		}
	}
	cout << "Програма підраховує кількість літер a, o, e (та всіх інших) в слові та
замінює їх відповідно на 1, 2, 3" << endl;
	//Цикл для виведення на друк рузультатів розрахунку для кожного слова,
	//які збережені у векторі
	for (vector<strVec>::iterator i = vect.begin(); i != vect.end(); i++)
	{
		sVec123 = *i;
		cout << sVec123.word << ", слово після заміни " << sVec123.word123
			<< ", кількість a=" << sVec123.count_a << ", кількість o="
			<< sVec123.count_b << ", кількість e=" << sVec123.count_c
			<< ", кількість інших літер n=" << sVec123.count_n << endl;
	}
	cout << endl << "Вивести результати роботи програми у текстовий файл? [y] [n]"
		<< endl; //Запит про необхідність запису результатів у зовнішній файл
	char out;
	cin.clear();
	cin >> out;
	for (auto &i : vect) //Серійний цикл for для виведення у зовнішній файл рузультатів
				//розрахунку для кожного слова, які збережені у векторі
	{
```

```
                        //Переривання роботи циклу for, якщо користувач обрав "n"
                        if (out == 'n') break;
                        sVec123 = i;
                        outfile << sVec123.word << ", слово після заміни " << sVec123.word123
                                << ", кількість a=" << sVec123.count_a << ", кількість o="
                                << sVec123.count_b << ", кількість e=" << sVec123.count_c
                                << ", кількість інших літер n=" << sVec123.count_n << endl;
                }
        }
        catch (const std::runtime_error err)            //Перехоплення виключень
        {
                cout << "Виникла помилка в процесі виконання програми." << endl;
                cout << err.what() << endl;
        }
        catch (const std::out_of_range err)             //Перехоплення виключень
        {
                cout << "Виникла помилка. Значення поза межами діапазону." << endl;
                cout << err.what() << endl;
        }
        cout << "Повторити введення? [y] [n]" << endl;  //Запит на запуск повторного розрахунку
        cin.clear();                                    //Очищення потоку введення
        cin >> n;
    } while (n == 'y');
    infile.close();                                     // закриваємо файл
    outfile.close();                                    // закриваємо файл
    return 0;
}
```

***Приклад об'явлення нової структури даних у файлу заголовку.***

```
//strVec.h      Файл заголовку з об'явленням нової структури даних

#ifndef STRVEC_H
#define STRVEC_H
#include <string>
struct strVec //Створення власної структури даних (класу)
{
        std::string word;
        std::string word123;
        unsigned count_a=0;
        unsigned count_b=0;
        unsigned count_c=0;
        unsigned count_n=0;
};
#endif
```

## Контрольні питання

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.

***Загальні вимоги.***

1) Створити консольний додаток з ім'ям виду *PrizvischeKP8*.
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* власну *структуру даних* structf, яка зберігатиме розраховане значення функції та значення аргументів, від яких вона залежить.
2) Створити *вектор*, типом значень якого буде створена *структура даних* vector<structf>.
3) Запрограмувати *функцію користувача* functf у зовнішньому файлі functf.cpp, яка розраховуватиме необхідне значення функції на певній ітерації.
4) В якості *параметрів функції* functf використати *параметри посилання*.
5) Записувати розраховане значення функції на кожній ітерації та аргументи функції, як окремий елемент створеного *вектору* типу vector<structf>.
6) Створити *функцію користувача* prntf у зовнішньому файлі functf.cpp, яка буде виводити на екран та у зовнішній текстовий файл елементи *вектору* (розраховане значення функції та її аргументи).
7) В якості *параметрів функції* prntf використати *константні посилання*.
8) Об'явити у функції prntf локальну *статичну змінну* типу *static int* для підрахунку ітерацій та передбачити виведення номеру ітерації на екран та у зовнішній файл разом зі значеннями функції.

9) Об'явити створені *функції користувача* functf та prntf у *файлу заголовку*. Підключити створений *файл заголовку* до файлів коду з функцією main та функціями functf і prntf.

10) Передбачити перехоплення помилки відкриття вихідного файлу для виведення результатів розрахунку за допомогою *оператору throw* та *блоку try catch*.

11) По завершенню розрахунків всіх значень функції запропонувати користувачу з застосуванням оператору *циклу while* вибір: завершити роботу програми чи запустити її на повторний розрахунок. Вихід з циклу організувати за допомогою *оператору переходу break*.

## Теоретичні відомості

*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, скільки 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. Говорят, 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 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.

***Загальні вимоги.***

1) Створити консольний додаток з ім'ям виду *PrizvischeKP9*.
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 знаходження заданого інтегралу методом Сімпсона чи за формулою трапецій (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 межами інтегрування (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 (коефіцієнти рівняння, межі інтегрування, точність інтегрування) для ініціалізації об'єкту класу за допомогою розширеного конструктора;

вводити лише 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 інтегралу 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.

Визначення класу 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.

## 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 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?

# 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.

## Допоміжна

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

## Інформаційні ресурси

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

# Додаток А

## Індивідуальні завдання

### Завдання до комп'ютерного практикуму №1

1. Обчислити об`єм, який займає 5 кг $CO_2$ при t = -9°C та p = 1,64 атм. (Дивись формулу (1)).

2. Обчислити молекулярну масу хімічної сполуки та масову частку кожного з хімічних елементів, які входять у склад ортоалюмінату кальцію $Ca(AlO_3)_2$.

   (Використати формулу: $N_i = \frac{n_i * A_i}{\sum n_i A_i} * 100\%$ ).

3. Обчислити молекулярну масу хімічної сполуки та масову частку кожного з хімічних елементів, які входять у склад метіоніна $CH_3S(CH_2)_2CH(NH_2)CO_2H$. (Брутто формула $C_5H_{11}NO_2S$).

4. Знайти брутто формулу хімічної сполуки, якщо відомо, що її молекулярна маса дорівнює M = 73, а елементний склад $X_C$ = 49%; $X_H$ = 10%; $X_N$ = 19%; $X_O$ = 22%. (Використати формулу $n_i = \frac{M * X_i}{A_i}$).

5. Який об`єм ацетилену теоретично можливо спалити у 1 м$^3$ повітря. Який об`єм $CO_2$ буде при цьому отримано. (Прийняти, що 1 м$^3$ повітря має 0,21 м$^3$ кисню).

6. Знайти масу 200 л хлору при t = 0°C та p = 101,3 кПа. (Дивись формулу (1)).

7. Знайти масу 1,8 л $H_2S$, якщо відомо, що об`єм газу вимірювали при t = 17°C та p = 98,64 кПа. (Дивись формулу (1)).

8. При t = 0°C отримали суміш 5 л метану, 10 л водню та 25 л кисню. Розрахувати концентрацію складових частин газової суміші (у об`ємних %, та г/л).

9. Згідно з умовою завдання 8 розрахувати концентрацію у г/л та % за масою.

10. Згідно з умовою завдання 8 розрахувати концентрацію у % за масою та молях на літр.

11. Згідно з умовою завдання 8 розрахувати концентрацію у об`ємних % та молях на літр.

12. Згідно з умовою завдання 8 розрахувати парціальний тиск компонентів, якщо тиск дорівнює 101,3 кПа.

13. Обчислити об'єм, який займає 7 г $CO_2$ при t = 7°C та p = 104 кПа. (Дивись формулу (1)).

14. Об'єм газу при $t_1$ = 23°C та $p_1$ = 103,3 кПа дорівнює 250 л. Знайти об'єм, який займає газ при $t_2$ = 0°C та $p_2$ = 101,3 кПа.

(Використати формулу $\frac{P_1*V_1}{T_1} = \frac{P_2*V_2}{T_2}$).

15. Розрахувати, скільки молекул вміщає 1 мл газу при t = 23°C та p=2,53 кПа. (Число Авогадро A = $6,02*10^{23}$).

16. Розрахувати, скільки молекул вміщає 1 мл газу при нормальних умовах (t = 0°C та p = 101,3 кПа). (Число Авогадро A = $6,02*10^{23}$).

17. Маса 1 л газу при нормальних умовах ($t_2$ = 0°C та $p_2$ = 101,3 кПа) дорівнює 1,25 г. Розрахувати мольну масу газу, та масу його молекули. (Число Авогадро A = $6,02*10^{23}$, дивись формулу (1)).

18. Концентрація гідроксильних іонів дорівнює $[OH^-]$ = $2,5*10^{-12}$ моль/л. Розрахувати pH розчину. (Пам`ятати, що $[OH^-][H^+] = K_w = 10^{-14}$.

19. Концентрація гідроксильних іонів дорівнює $[OH^-] = 9*10^{-9}$ моль/л. Розрахувати pH розчину. (Пам`ятати, що $[OH^-][H^+] = K_w = 10^{-14}$.

20. Знайти кількість молекул, яку вміщує 1 г $H_2SO$ та 1 г $HNO_3$. У якому з`єднанні (у скільки разів) молекул більше? (Число Авогадро A = $6,02*10^{23}$).

21. Скільки років потрібно, щоб перерахувати кількість молекул, яку вміщає 1 г води, якщо рахувати 1 молекулу в 1 секунду (1 рік = 365 днів).

22. Скільки років потрібно, щоб із заповненої гелієм ампули (умови нормальні, t = 0°C; p = 101,3 кПа) ємкістю 1 $мм^3$ повністю евакувати весь газ із швидкістю $10^6$ атомів у секунду. (Число Авогадро A = $6,02*10^{23}$).

23. Розрахувати рівнодіючу двох сил $F_1$ = 24 Н і $F_2$ = 38 Н, які діють на одну точку під кутом 48° одна до одної. (Дивись формулу 2.2)

24. Крапля, яка має заряд q = $1*10^{-10}$ кл, розділилася на дві частини, які на відстані R = $5*10^{-10}$ м взаємодіють з силою F = 1 Н. Знайти заряд кожної частини. ($q_1 + q_2 = q$; $F = 4ПE_0R$; E = $8,8541878*10^{-2}$ кл/(В*м).).

25. Ввести два числа A та B. Знайти число C, яке дорівнює B відсотків від числа A.

26. Обчислити площу трикутника та синуси всіх його кутів, якщо відомі сторони трикутника. (Дивись формулу 2.4 та 2.5).

27. Знайти висоти трикутника за його сторонами. (Дивись формули 2.3 та 2.4).

28. За двома сторонами та кутом між ними знайти третю сторону та площу трикутника. (Дивись формули 2.4 та 2.5).

29. Сторони трикутника дорівнюють 4 см, 5 см, 6 см. Знайти площу трикутника, сторони якого є медіанами заданого. (Дивись формули 2.4 та 2.5).

30. Основа рівнобедреного трикутника дорівнює A, а кут між рівними його сторонами – b. Знайти площу трикутника та тригонометричні функції всіх його кутів. (Дивись формулу 2.3 то 2.5).

## Завдання до комп'ютерного практикуму №2

1. Розробити алгоритм і програму для визначення залежності:

$$y=\begin{cases}\left|\sin x^2-b\cdot\cos^2 x^3\right| & при \quad x<0\\ \left|\lg x^2-5{,}1\right|+l^{-x^2+1} & при \quad 0\le x<1\\ tg^2(x-2)-2{,}5 & при \quad x\ge 1\end{cases}$$

де $x = a * \text{Sin}^2 t - b * \text{Cos}\, t^3$; $a = 1{,}3$; $b = 1{,}5$; $t = 1{,}8$.

2. Розробити алгоритм і програму для визначення залежності:

$$y=\begin{cases}\ln\left|(x-1{,}5)\cdot\sin x\right|+2\cdot ax^2 & при \quad x<1\\ tgx-b\cdot\cos^2 x^3 & при \quad 1\le x\le 2{,}1\\ l^{-x^2+1.5x}+\ln x & при \quad x>2{,}1\end{cases}$$

де $a = 1.2$; $b = 2{,}1$; $x = a * \ln t$; $t = 2.8$.

3. Розробити алгоритм і програму для визначення залежності:

$$y=\begin{cases}\ln\left|x-3\right|+1{,}4 & при\ x<0\\ tg(x-1)+a\cdot\cos^2 x^3 & при\ 0\le x<1\\ \ln(x^2+1)-e^{-2x} & при\ x\ge 1\end{cases}$$

де $x = a * \text{Sin}\, t - b * \text{Cos}(2t)$; $a = 2{,}5$; $b = 0{,}8$; $t = 2{,}93$.

4. Розробити алгоритм і програму для визначення залежності:

$$Z=\begin{cases}3 & при \quad x=1{,}5\\ \ln x & при - \quad x>2\\ \sqrt{|x|} & при \quad x\le -3\\ \left|x+tgx^2\right| & при \quad -3<x\le 2\end{cases}$$

де $x = e^{-\text{Sin}\, t} + \text{Cos}\, t$, $t = 1{,}37$.

5. Розробити алгоритм і програму для визначення залежності:

$$Z=\begin{cases}e^{-x}+tgx & при\ 2\le x\le 2{,}5\\ \ln x^2+b\cdot\ln x^4 & при \quad x>2\\ \cos^2 x+b\cdot\cos^3 x & при \quad x<2\end{cases}$$

де $x = a * \text{Sin}^3 t + 1.5$; $a = 1.2$; $b = 2.3$; $t = 0.8$.

6. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases}\ln\left|\sin^3 x^3\right|-1 & при \quad x\le 2\\ 10^{-2x^2+\sin x} & при \quad 2<x\le 2{,}3\\ \cos x+2\sin x^2 & при \quad x>2{,}3\end{cases}$$

де $x = a * t - \text{ctg}(b * t)$; $a = 0{,}2$; $b = -0{,}4$; $t = 4{,}7$.

7. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases}\lg(x^2-2x) & при \quad x<0\\ \ln(x+4) & при \quad 0\le x<10\\ \log_x 5-\log_5 x & при \quad x\ge 10\end{cases}$$

де $x = a * \text{Cos}^2 t - b * t$; $t = 1$; $a = 0{,}5$; $b = 6$.

8. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases}\left|tgx-\cos^3 2x\right| & при \quad x<0\\ x^2+x^3-5x^4 & при \quad 0\le x<5\\ x+\lg x & при \quad x>5\end{cases}$$

де $x = a * t + b$; $a = 5$; $t = 4$; $b = 0{,}1$.

9. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases}\left|x^2-3x\right| & при \quad x<0\\ t^2-4 & при \quad x=0\\ -x^3-x & при \quad x>0\end{cases}$$

де $x = \exp(t^4 + t + 2t - t)$; $t = 4$.

10. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases}\left|\sin x^2-\cos^2 x\right| & при \quad x<1\\ \cos 2x+\sin 2x & при \quad 1\le x<2\\ \left|x-5\right| & при \quad x\ge 2\end{cases}$$

де $x = t^2 + a * \text{Cos } t$; $a = 2$; $t = 0$.

11. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases}\left|\log_7 7x\right| & при \quad x<0\\ 5x+4 & при \quad 0\le x<15\\ \left|-x^{\sin x+3}\right| & при \quad x\ge 15\end{cases}$$

де $x = t^{\frac{a}{b}}$; $t = 4$; $a = 3$; $b = 2$.

12. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} e^{x^2} & при \quad x < -3 \\ |x| & при \quad -3 \le x < 3 \\ \sin x - \cos 2x & при \quad x \ge 3 \end{cases}$$

де $x = t^3 - a * t + b$; $a = 2$; $b = 4$; $t = 1$.

13. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} -e^{-x^2+2x} & при \quad x < 3 \\ \sin x^4 + ctg2x & при \quad 3 \le x < 4 \\ 5 & при \quad x \ge 4 \end{cases}$$

де $x = \text{Cos}(\ln t^a + b)$; $a = 0{,}3$; $b = 2{,}1$; $t = 1{,}7$.

14. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \sin|x| & при \quad x < 10 \\ 10^x + \lg 3x^2 & при \quad x = 10 \\ \cos|x| & при \quad x > 10 \end{cases}$$

де $x = a * \text{Sin } t^b - \text{Cos}(a * t)$; $a = 2$; $b = 3$; $t = 1{,}5$.

15. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} tg^2|x^2 - 4x + 3| & при \quad x < 2 \\ |2x + 5| & при \quad 2 \le x < 5 \\ 4 & при \quad x \ge 5 \end{cases}$$

де $x = 4t^3 - a * t + t^{-b/a}$; $a = 1{,}5$; $b = -2$; $t = -0{,}5$.

16. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \cos^2(x + 4) & при \quad x > 2 \\ |\sin x - e^x| & при \quad 2 \le x < 4 \\ \ln 3x & при \quad x \ge 4 \end{cases}$$

де $x = a * t^3 - b * t$; $a = 1{,}5$; $b = 3$; $t = 2{,}3$.

17. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases} e^{-x}+ctgx\cdot tg2x & при \quad x<-10 \\ |x^3| & при \quad -10\le x<-1 \\ x^4+10^x & при \quad x\ge -1 \end{cases}$$

де $x = a * t - b * t^2$; $a = 1,5$; $b = 3$; $t = 2,3$.

18. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases} \sin^4(x-5) & при \quad x<0 \\ x^3-\sqrt{x} & при \quad 0\le x<10 \\ e^{-x}+|x| & при \quad x\ge 10 \end{cases}$$

де $x=\dfrac{\tan t}{\sin t+\cos t}$; $t$ =1,2.

19. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases} \dfrac{1}{x}+\dfrac{1}{3x^2} & при \quad x<5 \\ 7x^{\ln x} & при \quad 5\le x<7 \\ 2\sin x^4+\cos x & при \quad x\ge 7 \end{cases}$$

де $x = a * \cos t^2 - b * t$; $a = 1$; $b = 0,2$; $t = 0,4$.

20. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases} e^{-2x}+x & при \quad x<0 \\ x^4+5x & при \quad x=0 \\ |\sin x+tgx| & при \quad x>0 \end{cases}$$

де $x = b * t^2 + a$; $a = -3$; $b = 4$; $t = 3,5$.

21. Розробити алгоритм і програму для визначення залежності:

$$Y=\begin{cases} 5^{tgx}-4 & при \quad x<2 \\ \dfrac{1}{2\sin x+3} & при \quad 2\le x<3 \\ (2\sin x)^x & при \quad x\ge 3 \end{cases}$$

де $x = 2,5t^2 - a / t$; $a = 5$; $t = 7,8$.

22. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \cos 5x - x^{2,3} & при \quad x < -2 \\ \left|\ln x^4 - 5\right| & при \quad -2 \le x < 2 \\ \dfrac{1}{tgx + ctg2x} & при \quad x \ge 2 \end{cases}$$

де $x = a * \text{Sin } t^b - a$; $a = 0,2$; $b = 0,4$; $t = 0,3$.

23. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} 10^{\sin x - \cos 2x} & при \quad x < 3 \\ 5 & при \quad 3 \le x < 9 \\ \left|x^6 - 7\sqrt{x}\right| & при \quad x \ge 9 \end{cases}$$

де $x = \ln t^a + b$; $a = 3$; $b = 0,3$; $t = 2,7$.

24. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} x^2 + 4 & при \quad x < -2 \\ \left|\sin x - 2^{-x}\right| & при \quad -2 \le x < 10 \\ \log_5 x^2 & при \quad x \ge 10 \end{cases}$$

де $x = t^a + b * t^b$; $a = 0,3$; $b = 0,4$; $t = 0,2$.

25. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \lg(3x - x^2) & при \quad 0 < x < 2 \\ \sin(3x^3 + 4) & при \quad 2 \le x < 3 \\ \dfrac{1}{x^4 + 5} & при \quad x \ge 3 \end{cases}$$

де $x = t^{a * b - b}$; $a = 0,3$; $b = 1,1$; $t = 0,7$.

26. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \left|\sin^2 x + \cos^3 2x\right| & при \quad x < 2 \\ \dfrac{x^3 - 5x^4}{x^2} & при \quad 2 \le x < 4 \\ \dfrac{a \cdot x + \lg x}{b} & при \quad x \ge 4 \end{cases}$$

де $x = a * t + b$; $a = 1,8$; $t = 2$; $b = 3,4$.

27. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \left|\sin^2 x - \cos x^2\right| & при \quad x < 0{,}5 \\ \cos a \cdot x + \sin a \cdot x & при \quad 0{,}5 \le x < 3{,}1 \\ \left|x - a\right| & при \quad x \ge 3{,}1 \end{cases}$$

де $x = t^2 + a * \text{Sin } t$; $a = 2$; $t = 0.5$.

28. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} e^{-bx} + tg^2 x / x & при\ 1{,}5 < x < 3{,}5 \\ \lg x^2 + b \cdot \lg x^4 & при\ x > 3{,}5 \\ \cos^3 x + b \cdot \cos a \cdot x & при\ x < 1{,}5 \\ t \cdot \sin b \cdot x / a & при\ x = 3{,}5 \end{cases}$$

де $x = a * \text{Sin}^2 t + 1.9$; $a = 2.7$; $b = 1.5$; $t = 0.8$.

29. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} \lg(x - x^3) & при\ 0 < x < 5 \\ \cos(2x^3 + 3) & при\ 5 \le x < 7 \\ \dfrac{1}{x^2 + 3} & при\ x \ge 7 \end{cases}$$

де $x = t^{a * b - b}$; $a = 0{,}5$; $b = 1{,}5$; $t = 0{,}9$.

30. Розробити алгоритм і програму для визначення залежності:

$$Y = \begin{cases} e^{x^2 + 3x} & при \quad x < 3 \\ \cos x^3 + tg3x & при \quad 3 \le x < 4 \\ \sin 5x & при \quad x \ge 4 \end{cases}$$

де $x = \sin(\ln t^a + b)$; $a = 0{,}2$; $b = 2{,}5$; $t = 1{,}6$.

## Завдання до комп'ютерного практикуму №3

***Частина 1.***

1. Розрахувати в'язкість аміаку в інтервалі температур від 12 до 18°С через 1°С за формулою Сатерленда

$$N = N_0 \frac{273{,}15 + C}{T \cdot C} \cdot (T/273{,}15)^n$$

При умові, що множник n визначається так:

$$n = A - B \cdot (t - 273{,}15) \cdot D \cdot 10^{-7} \cdot (t - 273{,}15)^2$$

де *A* = 1,06; *B* = 1,04; *D* = 0; *C* = 503; *N* = 91,6 * $10^{-7}$ Па*с.

2. Розрахувати ізобарну теплоємність $CO_2$ в інтервалі температур від 30 до 150°С через 10°С за формулою:

$$C_p = E + F \cdot (T/100) + G \cdot (T/100) + H \cdot (T/100) + N \cdot (T/100)$$

де *E* = 0,81513; *F* = 9,8454· $10^{-2}$; *G* = -9,4747· $10^{-3}$; *H* = 36,006· $10^{-5}$; *N* = -0,0567.

3. Використавши формулу завдання 2 при *E* = 2,0183; *F* = 158,072 * $10^{-2}$; *G* = 273,61 * $10^{-3}$; *H* = -9380,9 * $10^{-5}$; *N* = -1,9986, розрахувати ізобарну теплоємність аміаку в інтервалі температур від 80 до 150°С через 10°С.

4. За умови завдання 2 розрахувати ізохорну теплоємність $CO_2$. Формула зв'язку має вигляд: $C_v = C_p - R/M$ (*C* – молекулярна маса газу; *R* – універсальна газова стала).

5. За умови завдання 2 розрахувати ізохорну теплоємність аміаку. Формула зв'язку має вигляд: $C_v = C_p - R/M$ (C – молекулярна маса газу; R – універсальна газова стала).

6. Розрахувати в'язкість води в інтервалі температур від 16 до 27°С через 1°С за формулою:

$$N_o = A(B + t)^n,$$

де *A* = 0,59849, *B* = 43, 252, *n* = -1,5423.

7. Розрахувати теплопровідність аміаку в інтервалі температур від 16 до 27°С через 1°С за формулою:

$$L = A \frac{(T_K + c)}{(T + C)} \cdot \frac{T^{\frac{3}{2}}}{T_K^{\frac{3}{2}} M^{\frac{5}{6}}},$$

де A = 4,3643 * $10^{-3}$; C = 503; $T_K$ = 239,73 К; *M* – молекулярна маса газу.

8. Розрахувати густину повітря в інтервалі температур від 45 до 54°С за формулою:

$$R = \frac{1{,}293 \cdot P}{(1 + 0{,}00367 \cdot t) \cdot 760}, \text{кг/м}^3$$

де *P* – тиск, який дорівнює 105,2 кПа.

9. Розрахувати швидкість осадження кулеподібної частинки радіусом r ($r = 0{,}2 \cdot 10^{-3}$ м, $r = 0{,}3 \cdot 10^{-3}$ м … $r = 0{,}7 \cdot 10^{-3}$ м) у газі. Використати формулу:

$$W_{oc} = \frac{d^2 \cdot R \cdot g}{18 \cdot M_c},$$

де $g$ = 9,81 м$^2$/с, $R$ = 1684 кг/м$^3$; $M_c$ = 5,2 * $10^{-4}$ Н * с/м$^2$.

10. Розрахувати час процесу фільтрування суспензії об'ємом V ($V = 50 \cdot 10^{-6}$, $100 \cdot 10^{-6}$, $150 \cdot 10^{-6}$, $200 \cdot 10^{-6}$, $250 \cdot 10^{-6}$ м$^3$) за формулою:

$$t = \frac{M \cdot r_0 \cdot x_0}{2p} \cdot \frac{V^2}{S^2} + \frac{M \cdot R}{P} \cdot \frac{V}{S},$$

де $M = 5{,}17 \cdot 10^{-9}$ Н * с/м$^3$; $r = 4{,}15 * 10^{12}$ м$^{-2}$; $R = 2{,}3 * 10^{10}$ м$^{-1}$; $x_0 = 0{,}178$ м$^3$/м$^3$; $P = 0{,}6 \cdot 10^5$ Н/м$^2$; $S = 1$ м.

11. Розрахувати витрату потужності на перемішування для пропелерної мішалки діаметром $d$ = [2,6; 2,8; 3,0; 3,2 м]. Використати формулу

$$N = K_N \cdot n^3 \cdot R_C \cdot d_m^5,$$

де $K_N$ = 0,32; т = 3,96 об/с; $R_C$ = 1200 кг/м$^3$.

12. Розрахувати коефіцієнт теплопровідності L нітробензолу в інтервалі температур від 40 до 110°С через 10°С. Використати формулу

$$L_t = L_{30} \cdot [1 - E \cdot (t - 30)],$$

де $L_{30} = \mathrm{A}_1 \cdot \mathrm{c} \cdot R \cdot \sqrt{\frac{R}{M}}$, $\mathrm{A}_1 = 4{,}22 \cdot 10^{-2}$; с=1,38· $10^3$ Дж/(кг*град); $R$ = 1200 кг/м$^3$; $M$ = 123; $E$ =1· $10^{-3}$ град$^{-1}$.

13. Розрахувати значення густини та в'язкості сірчаної кислоти в інтервалі температур від 10 до 80°С через 10°С. Використати формулу

$$R = 1894{,}8 - 0{,}909 \cdot t; \qquad M = 1{,}406 \cdot 10^{-3} + 5{,}0087 \cdot 10^{-1}/t$$

14. Розрахувати опір шару осадка при фільтруванні суспензії з діаметром частинок $d_{\text{ср}}$ = (0,1; 0,15; 0,20; 0,25) * $10^{-3}$ м за формулою:

$$r_{0_{11}} = A \cdot d_{cp}^B,$$

де $A$ = 9,8373*10; $B$ = -0,3224.

15. Для десяти значень $x$ перевірити, чи є рівняння

$$sin^6 x + cos^6 x = 1 - \frac{3}{4} \cdot sin^2 2x$$

тотожністю.

16. Виконати завдання 16 для формули:
$\tan x + \tan 2x - \tan 3x = -\tan x \cdot \tan 2x \cdot \tan 3x$.

17. Виконати завдання 16 для формули:

$$\sin 2nA + \sin 2nB + \sin 2nG = (-1)^{n+1} \cdot 4 \cdot \sin nA \cdot \sin nB \cdot \sin nG,$$

де n = 1, 2, 3, 4…

18. Виконати 5-7 перевірок рівняння

$$\sin A + \sin B + \sin G = 4 \cdot \sin\frac{A+B}{2} \cdot \sin\frac{B+G}{2} \cdot \sin\frac{A+G}{2}$$

на тотожність, якщо *A*, *B* і *G* – кути трикутника.

19. Розрахувати залежність продуктивності вакуумфільтру *W* (кг/(час*м$^3$)) від величини вакууму *P* (Н/м$^2$), який визначається від 50 до 100 мм рт. ст. з кроком *P* = 25 мм рт. ст. для трьох розмірів *d* = 0,25; 0,30; 0,35 мм, якщо:

$$W = 7056 \cdot P^{0,5} \cdot d^2$$

20. Розрахувати тиск пару хлористого метилену *P* (та $\ln P$) в інтервалі температур 20…40°С через 5°С, якщо

$$\ln P = -\frac{A}{T} + B \cdot \ln T + C,$$

де *A* = 1995,6; *B* = -3,4426; *T* = *t* + 273; *C* = 8,321; 8,621; 8,921.

21. Розрахувати швидкість *R* хімічної реакції *A* -> *B* в діапазоні температур *t* = 300…400К з кроком 20 К, якщо

$$R = К_0 \cdot \exp\left(-\frac{E}{R} \cdot T\right) \cdot C_0,$$

де $К_0$ = 2,05 * с$^{-1}$; *R* = 1,98725 кал/моль*град; *E* = 20100 кал/моль; $C_0$ = (2; 4; 6) моль/м$^3$.

22. Розрахувати, який об'єм займе 10 г азоту при тиску *P* = 1,5; 3,0; 4,5 атм і зміні температури від 200 до 400 К з кроком 10 К.
(Використати формулу 1; *R* = 0,082057 л*атм/моль*град)

23. Розрахувати значення константи швидкості хімічної реакції

$$K = A \cdot \exp\left(\frac{-5,29 \cdot 10^4}{R \cdot T}\right)$$

В інтервалі температур 600 … 800 через 50°С, якщо А = 7,5· $10^{-5}$; 8,0· $10^{-5}$; 8,5· $10^{-5}$.

24. Розрахувати значення енергії Гіббса

$$G = -R \cdot T \cdot \ln K$$

хімічної реакції в інтервалі температур від 400 до 500 К з кроком 20 К, якщо R = 1,3143 Дж/моль*К, К = 9,137; 9,237; 9,337.

25. Розрахувати об'єм кульового сегменту

$$V = \frac{П \cdot h}{6} \cdot (h^2 + 3 \cdot r^2),$$

якщо *h* змінюється від 0,1 до 0,2 м з кроком 0,05 м, для *r* = 0,5; 1,0; 1,5 м.

26. Обчислити та запам'ятати значення:

$$y_{ij} = A_i^{2,2} + 1{,}5 \cdot A_i \cdot B_i \cdot C_j - B_i^{1,5} \cdot \sqrt{A_i + 2B_i}; \qquad A_i = 1{,}8 \cdot \cos^3 x_i; \qquad B_i = e^{2/x_i} + 2{,}6.$$

а потім знайти суму всіх отриманих значень функції $y_{ij}$.
Прийняти $x_i$ = 2,2; 1,9; 1,7; 1,4; 1,2; $C_j$ = 2,31; 2,86; 3,57.

27. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в $n$ різних точках $(x_i, y_i)$, а потім знайти суму всіх отриманих значень функції. Прийняти $n > 10$; $x_{i+1} = x_i + \delta x$ ($x_0 = 2.8$; $\delta x = 2.1$); $y_{i+1} = y_i + \delta y$ ($y_0 = 1.7$; $\delta y = 0.3$).

$$F(x_i, y_i) = [\ln(x_i^2) - \ln(y_i^2)] \frac{\sqrt[3]{x_i - y_i}}{\operatorname{ctg}(x_i)}.$$

28. Обчислити значення:

$$y_i = a_i^{2b_i} + b_i^{3a_i}; \qquad a_i = e^{2{,}5/x_i} + 3x_i \cdot C; \qquad b_i = \sin^2 x_i + x_i^{0{,}5}.$$

Прийняти $x_i$ = 0,3; 0,7; 0.9; 1,1; 1,2; 1.4; 1,7; 1,8; 1,9; 2.2; 2.5; 2.7; C=2,15;.

29. Розрахувати значення тригонометричних функцій sin(A), cos(A) і ctg(A) при зміні A від 30 до 150° з кроком 15°.

30. Знайти перші десять та обчислити суму перших трьох, п'яти та дев'яти членів ряду, у якому

$$A_n = \frac{2n! - (-1)^n \cdot 2^{n-1}}{2^{n+1} + n!}$$

***Частина 2.***

1. $S \approx \frac{1}{x}+1+\frac{x}{2!}+\frac{x^2}{3!}+\text{K} = \frac{e^x}{x} \qquad |x|<1$
2. $S \approx 1+\frac{1}{4}x+\frac{1}{4}\cdot\frac{5}{8}\cdot x^2+\frac{1}{4}\cdot\frac{5}{8}\cdot\frac{9}{12}x^3+\ldots=(1-x)^{-1/4} \quad |x|<1$
3. $S \approx 1+\frac{1}{2}\cdot x+\frac{1\cdot 3}{2\cdot 4}\cdot x^2+\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\cdot x^3\text{K} = (1-x)^{-\frac{1}{2}} |x<1|$
4. $S \approx -x-\frac{x^2}{2}-\frac{x^3}{3}-\frac{x^4}{4}-\cdots = \ln(1-x) \quad -1\le x<1$
5. $S \approx x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+\ldots=arctg(x) \quad |x|\le 1$
6. $S \approx 1-x+x^2-x^3+\text{K} = \frac{1}{1+x} \qquad |x|<1$
7. $S \approx 1+\frac{1}{3}x+\frac{1\cdot 4}{3\cdot 6}x^2+\frac{1\cdot 4\cdot 7}{3\cdot 6\cdot 9}x^3+\ldots=(1-x)^{-1/3} \quad |x|<1$
8. $S \approx 2\left(x+\frac{x^3}{3}+\frac{x^5}{5}+\frac{x^7}{7}+\text{K}\right)=\ln\frac{1+x}{1-x} \qquad |x|<1$
9. $S \approx x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots=\ln(1+x) \quad -1<x\le 1$
10. $S \approx 1+x+x^2+x^3+\ldots=\frac{1}{1-x} \qquad |x|<1$
11. $S \approx 1-x^2+x^4-x^6+x^8-\text{K} = \frac{1}{1+x^2} \qquad |x|<1$
12. $S \approx 1+\frac{1}{x}+\frac{1}{2!x^2}+\frac{1}{3!x^3}+\ldots=e^{1/x} \quad |x|\le 1$
13. $S \approx 1-2x+3x^2-4x^3+5x^4-\text{K}=\frac{1}{(1+x)^2} \quad |x|<1$
14. $S \approx 1-\frac{1}{4}\cdot x+\frac{1}{4}\cdot\frac{5}{8}\cdot x^2-\frac{1}{4}\cdot\frac{5}{8}\cdot\frac{9}{12}\cdot x^3+\text{K} = (1+x)^{-1/4} \qquad |x|<1$
15. $S = x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots=\sin(x) \quad |x|\le\infty$
16. $S \approx 1-\frac{5}{2}x+\frac{5\cdot 7}{2\cdot 4}x^2-\frac{5\cdot 7\cdot 9}{2\cdot 4\cdot 6}x^3+\ldots=(1+x)^{\frac{-5}{2}} \quad |x|<1$
17. $S = 1-x+\frac{x^2}{2!}-\frac{x^3}{3!}+\ldots=e^{-x} \qquad |x|\le\infty$
18. $-S \approx 1-\frac{1}{3}x+\frac{1}{3}\cdot\frac{4}{6}x^2-\frac{1}{3}\cdot\frac{4}{6}\cdot\frac{7}{9}x^3+\ldots=(1+x)^{-1/3} \quad |x|<1$
19. $-S \approx 1-\frac{1}{2}\cdot x+\frac{1\cdot 3}{2\cdot 4}\cdot x^2-\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\cdot x^3+\ldots=(1+x)^{-\frac{1}{2}} \quad |x|<1$

20.- $S \approx 1-\frac{3}{2}\cdot x+\frac{3\cdot 5}{2\cdot 4}\cdot x^2-\frac{3\cdot 5\cdot 7}{2\cdot 4\cdot 6}\cdot x^3+\mathrm{K}=(1+x)^{-3/2}$ $\quad |x|<1$

21. $S \approx 1-\frac{2\cdot 3}{2}x+\frac{3\cdot 4}{2}x^2-\frac{4\cdot 5}{2}x^3+\cdots=\frac{1}{(1+x)^3}$ $\quad |x|<1$

22. $S \approx 1+\frac{3}{2}\cdot x+\frac{3\cdot 5}{2\cdot 4}\cdot x^2+\frac{3\cdot 5\cdot 7}{2\cdot 4\cdot 6}\cdot x^3+\mathrm{K}=(1-x)^{-3/2}$ $\quad |x|<1$

23. $S \approx \frac{1}{x}-\left(\frac{x}{3}+\frac{x^3}{45}+\frac{2x^5}{945}+\frac{x^7}{4725}+\ldots\right)=ctg(x)$ $\quad (0<|x|<\pi)$

24.- $S \approx -\frac{x^2}{2}-\frac{x^4}{12}-\frac{x^6}{45}-\frac{17x^8}{2520}-\ldots=\ln|\cos(x)|$ $\quad (|x|<\frac{\pi}{2})$

25. $S \approx x+\frac{x^3}{3!}+\frac{x^5}{5!}+\frac{x^7}{7!}+\mathrm{K}=sh(x)$ $\quad |x|\le\infty$ $\quad \left[sh(x)=\frac{e^x-e^{-x}}{2}\right]$

26. $S=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots=\cos(x)$ $\quad |x|\le\infty$

27. $S \approx 1+\frac{x^2}{2!}+\frac{x^4}{4!}+\frac{x^6}{6!}+\mathrm{K}=ch(x)$ $\quad |x|\le\infty$ $\quad \left[ch(x)=\frac{e^x+e^{-x}}{2}\right]$

28. Розрахувати *k* членів ряду $S \approx x+\frac{x+1}{2x}+\frac{x+2}{3x^2}+\mathrm{K}$ $\quad |x|\le\infty$

29. Розрахувати *k* членів ряду $S \approx 1+\frac{1}{x}+\frac{1-x}{2!x^2}+\frac{1-2x}{3!x^3}+\frac{1-3x}{4!x^4}\ldots$ $\quad |x|\le 1$

30. Розрахувати *k* членів ряду $S \approx 1-\frac{x^2}{x-2!}+\frac{x^4}{x-3!}-\frac{x^6}{x-4!}+\frac{x^8}{x-5!}-\mathrm{K}$ $\quad |x|<1$

## Завдання до комп'ютерного практикуму №4

***Частина 1.***

1. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в ***n*** різних точках $(x_i, y_i)$, де ***n*** > 16; $x_i$ та $y_i$ – довільні додатні числа;

$$F(x_i, y_i) = \sin^2(x_i^{\,2} \cdot y_i) + \cos^3(x_i \cdot y_i^{\,3}) - \sqrt{\lg^3 x_i^{\,2}};$$

2. Визначити втрату тиску P охолоджуючої води, яка протікає через свинцевий змійовик, який має 40 витків діаметром D = 0.5 м. Внутрішній діаметр свинцевої труби d = 0,012 м. Середня температура охолоджуючої води 20°C, а середня температура внутрішньої стінки труби 40 °C. Витрата охолоджуючої води G = 0,1 кг/с.

$$P = L \cdot \frac{l}{d} \cdot \frac{W^2}{2} \cdot p \cdot f \cdot f1,$$

де L = 0.0335 – коефіцієнт тертя;
l = 3.14*D*n – загальна довжина труби (n = 40), м;
$W = \frac{4 \cdot G}{3.14 \cdot p \cdot d^2}$ - швидкість руху рідини, м/с (з = 998 кг/м$^3$ – щільність води при 20°C);
f, f1 – поправкові множини для неізотермічного потоку рідини у непрямих трубах (f = 0.85; f1 = 1+ 3,54d/D).

3. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, де *n* > 16; $x_i$ – довільні додатні числа; $y_i = \lg^2 x_i^3 + (i-1)/10$

$$F(x_i, y_i) = \operatorname{tg}^2(x_i - y_i) \cdot (x_i^{\,2} + 0{,}5 \cdot y_i).$$

4. Знайти коефіцієнт тепловіддачі L при конденсації пару на поверхні труби довжиною H-2м, яка утворює з горизонтальною поверхнею кут b= $\frac{\Pi}{3}$. Температура стінки $t_{ст} = 210\ ^0C$, температура пари $t\ \ = 220\ ^0C$.

$$L_b = L\sqrt{\sin b},$$

де L – коефіцієнт тепловіддачі для вертикальної труби, який розраховується за формулою:

$$L = 3 \cdot 10^{-3} \sqrt{\frac{H(t - t_{ст}) \cdot l^3 \cdot p^2 \cdot g}{r \cdot m^3}}$$

тут l = 0,652 Вт/(м*К) – теплопровідність конденсату;
p = 846 кг/м$^3$ - щільність конденсату;
g = 9,81 м/с$^2$ - прискорення вільного падіння;
r = 1,88*10$^6$ Дж/кг – теплота пароутворення;
m = 1,29*10$^{-4}$ н*с/м$^2$ - динамічна в'язкість конденсату.

5. Обчислити значення:

$$y_{ij} = z_i + \sqrt{z_i^2 + a_j^3 \cdot z_i^{0,3}}; \qquad z_i = e^{-5 \cdot x_i}.$$

Прийняти $x_i$ = 0,1; 0,4; 0,8; 1,2; $a_j$ = 0,2; 0,5; 0,9.

6. Розрахувати константу швидкості реакції між йодистим метилом і диметил-р-толундіном у розчині нітробензолу.

$$\frac{\frac{x+x_p}{\text{a}}}{a\cdot t\cdot\left[1-\left(\frac{x_p}{a}\right)^2\right]}\cdot\ln\left|\frac{\frac{x}{a}\left(1-\frac{x+x_p}{a^2}\right)-1}{a^2\cdot\left(\frac{x_p}{a}-\frac{x}{a}\right)}\right|$$

де a = 0,06 – початкова концентрація;
x = 0,01246 - рівноважна концентрація;
$x_p$ = 0,00675 – поточна концентрація;
$t = 12$.

7. Обчислити значення:

$$y_{ij} = (z_i^{\,2} + 0{,}33)/\text{tg}z_i - b\cdot\sqrt{\left|z_i^{\,2} - a_j\right|};\ z_i = e^{-1/x_i} + x_i^{\,2}.$$

Прийняти $x_i$ = 1,3; 1,5; 1,7; 1,9; $a_j$ = 0,33; 1,12; 3,72; 5,16; b=3.68.

8. Розрахувати в'язкість діоксину сірки при 300 $^0C$ і атмосферному тиску, використавши формулу:

$$\mu = 6{,}3\cdot 10^{-4}\cdot\frac{\text{м}^{1/2}\cdot \text{P}_{\text{кр}}^{2/3}}{\text{T}_{\text{кр}}^{1/6}}\cdot\frac{\text{T}_{\text{прив}}^{3/2}}{\text{T}_{\text{прив}}\cdot 0{,}8}$$

де м = 64 – молекулярна вага $SO_2$;
$\text{T}_{\text{кр}}$ = 430 $^0C$ - критична температура;
$\text{P}_{\text{кр}}$ = 77,7 атм – критичний тиск;
$\text{T}_{\text{прив}}$ = (300 + 273)/430 – приведена температура.
Надрукувати значення $\mu$ та $\sqrt{\mu}$.

9. Обчислити значення:

$$y_{ij} = z_i^3 - (z_i + a_j\cdot b)/(a_j^{\,0,4} + z_i^2);\ z_i = e^{-1/x_i} + x_i\cdot b\cdot\cos x_i.$$

Потім знайти суму всіх отриманих значень функції $y_{ij}$.
Прийняти $x_i$ = 1,3; 1,5; 1,8; 2,0; $a_j$ = 0,7; 0,85; 0,9; b = 5,75.

10. Рідина витікає із прямої посудини 1 до посудини 2 через круглий отвір $d$ = 2 см = 0,02 м. Різниця рівнів води в початковий момент $H$ = 2м. Площина поперечних перерізів посудини $S_1$ = 8м$^2$, $S_2$ = 6м$^2$. Знайти час, необхідний для того, щоб різниця рівнів зменшилася до 1м, для цього використати формулу:

$$t = \frac{8\cdot S_1\cdot S_2}{c\cdot 3.14\cdot d^2(S_1+S_2)\cdot\sqrt{2\cdot g}}\left(\sqrt{H_0}-\sqrt{H}\right)$$

де $c$ = 0,62 – коефіцієнт витрат; $g$ = 9.81
надрукувати значення $t$, $e^c$, $t^2$, $sin^2 d$.

11. Обчислити та запам'ятати значення $z_i$ = F($x_i$, $y_i$) в $n$ різних точках ($x_i$, $y_i$), а потім знайти добуток всіх отриманих значень функції. Прийняти $n$ > 16; $x_{i+1} = x_i + \delta x$ ($x_0$ = 1.3; $\delta x$ = 0.2); $y_i = x_i^2$;

$$\text{F}(x_i, y_i) = \lg^2(x_i\cdot y_i) + \frac{\sqrt{x_i^{\,2} + y_i^{\,2}}}{x_i + y_i};$$

12. Розрахувати коефіцієнт ефективності, використовуючи формулу:

$$\Phi_s = \frac{r_s}{3}\sqrt{\frac{W_p \cdot p_k}{n \cdot D_0 \cdot (C_s - C_0)}}$$

де $r_s = 0{,}317$;
$W_p$ = 4,85*$10^{-4}$;
$p_k$ = 1;
$n = 0{,}93$;
$D_0 = 0{,}08$;
$C_s = 1{,}6 \cdot 10^{-4}$;
$C_0 = 0$.
Надрукувати значення $\Phi_s, e^n, sin_s r$.

13. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках ($x_i$, $y_i$), де *n* > 15; $x_{i+1} = x_i + \delta x$ ($x_0$ = 1.2; $\delta x$ = 0.3); $y_i$ – довільні додатні числа;

$$F(x_i, y_i) = e^{-1/x} + x^y - y^x$$

14. Визначити константу швидкості реакції окислення оксиду азоту, використовуючи формулу:

$$K = \frac{1}{t} \cdot \frac{1}{(b-a)^2} \cdot \left[\frac{(b-a) \cdot x}{(a-x) \cdot a} + \ln\frac{(a-x) \cdot b}{(b-x) \cdot a}\right]$$

де b=7,06;
a=5,4;
t=11;
x=2,25.

15. Знайти діаметр корпусу випарювального апарату з природною циркуляцією розчину, використовуючи формулу:

$$D = \sqrt{\frac{1{,}27 \cdot n \cdot t^2 \cdot \sin L}{f} + (d \cdot 2 \cdot t)^2}$$

де n=1210 – кількість трубок;
t=0,05м;
d=0,4м;
f=0,85м;
L=1,047рад.

16. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках ($x_i$, $y_i$), а потім знайти суму перших п'яти значень та добуток двох останніх.

$$F(x_i, y_i) = e^{-2/x} + 0{,}8 \cdot x^y - 1{,}7 \cdot y^x.$$

Прийняти *n* > 18; $x_{i+1} = x_i + \delta x$ ($x_0$ = 1.2; $\delta x$ = 0.55); $y_i$ – довільні додатні числа;

17. Знайти коефіцієнт дифузії молекул ізобутану через шар зернин оксиду алюмінію, використовуючи формулу:

$$D = \frac{-R \cdot T \cdot r \cdot g_a}{0{,}023 \cdot P \cdot (x_2 - x_1)}$$

де R = 82,298 – універсальна газова стала;
T = 0,31;
r = 0,5;
$g_a$ = 0,08*$10^{-6}$;
P = 5;
$x_2$ = 0;
$x_1$ = 1.
Надрукувати значення D, $\sqrt{D}$, $e^T$.

18. Обчислити значення:

$$y_i = a_i^{b_i} + b_i^{a_i}; \qquad a_i = e^{-1/x_i} + x_i \cdot C; \qquad b_i = \cos^2 x_i + x_i^{0,3}.$$

Прийняти $x_i$ = 0,5; 0,8; 0.9; 1,2; 1,3; 1.5; 1,7; 1,8; 1,9; 2.1; 2.3; 2.4; C=1,86.

19. Визначити коефіцієнт дифузії двооксиду сірки у повітрі $t = 20\ ^0C$ та P=1атм. Використати формулу:

$$D = 4{,}3 \cdot 10^{-3} \cdot \frac{T^{3/2}}{P \cdot (V_A^{1/3} - V_B^{1/3})^2} \cdot \sqrt{\frac{1}{M_A} + \frac{1}{M_B}}$$

де $V_A - V_{SO_2}$ = 44,8 $см^3/моль$;
$V_B - V_{ВОЗ}$ = 29,9 $см^3/моль$;
T = 273 +t;
$M_A$ = 64;
$M_B$ = 28,95.

20. Обчислити та запам'ятати значення $z_i$ = F($x_i$, $y_i$) в *n* різних точках ($x_i$, $y_i$), де *n* > 18; $x_i$ – довільні додатні числа; $y_i$ – довільні від'ємні числа.

$$F(x_i, y_i) = 12 \cdot x_i^{0.65} - (1{,}9 \cdot x_i^2 - 0{,}85 \cdot y_i^{0.8}) \cdot tg^2(x_i) / y_i.$$

21. Визначити теплопровідність рідкого гептану при температурі $t = 20\ ^0C$, використовуючи формулу:

$$L = A \cdot C_M \cdot y_{\text{відн}} \cdot \sqrt[3]{\frac{y_{\text{отн}}}{M}}$$

де A = 1,52;
$C_M$ = 51,9 + 0,142 * (t + 30), кал/(моль * град) – мольна теплоємність;
$y_{\text{відн}}$ = 0,675 – відносна питома вага;
M = 100;
C = $C_M/100$ - питома теплоємність.

22. Обчислити та запам'ятати значення:

$$y_{ij} = (A_i \cdot \sin B_j + B_j \cdot \cos A_i) \cdot (A_i^{0,5} + \sqrt[3]{B_j});$$

$$A_i = \cos^2 x_i + x_i^3; \; B_j = \sin^2 a_j + a_j^{0,5}.$$

Прийняти $x_i$ = 1,2; 0,4; 1,3; 0,8; $a_j$ = 0,6; 0,9; 1,9.

23. Знайти тривалість адсорбції суміші парів етилового спирту та діетилового ефіру шаром активованого вугілля висотою H = 1м, використовуючи формулу:

$$t = \frac{a_0}{w \cdot C_0} \cdot \left[H - \frac{w}{b} \cdot (\ln \frac{C_0}{C} - 1)\right]$$

де $a_0$ = 100кг/м$^3$ - кількість адсорбції речовини, зрівноваженої з концентрацією потоку;
$w = 12$м/хв - швидкість парогазового потоку;
$C_0$ = 0,072кг/м$^3$ – початкова концентрація;
C = 0,001кг/м$^3$ – середня концентрація на виході;
$b$ = 60 * 7,4 хв$^{-1}$ - коефіцієнт масо передачі.

24. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках ($x_i$, $y_i$), а потім знайти суму перших п'яти значень та добуток двох останніх. Прийняти *n* > 16; $x_{i+1} = x_i + \delta x$ ($x_0$ = 1.5; $\delta x$ = 0.65); $y_i$ – довільні додатні числа;

$$F(x_i, y_i) = 0{,}8 \cdot x^y - 1{,}7 \cdot y^x + \ln \frac{x}{y}.$$

25. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в ***n*** різних точках ($x_i$, $y_i$), де ***n*** > 16; $x = x_i * \delta x$ (x = 1.8; $\delta x$ = 1.2); $y_i$ – довільні додатні числа ($y_i$ < 1).

$$F(x_i, y_i) = \frac{\sqrt{1 - y_i}}{x_i + \operatorname{tg} x_i} \cdot (1 + e^{-x_i \cdot y_i}).$$

26. Обчислити значення:

$$y_{ij} = z^{0,3} + z_i \cdot a_j; \qquad z_i = x_i^2 + \ln x_i.$$

Прийняти $x_i$ = 1,2; 1,6; 1,9; 1,95; $a_j$ = 0,9; 0,95; 0,99; b=1.68.

27. Обчислити значення:

$$y_{ij} = z_i + (a_j \cdot z_i \cdot b + b^{0,3})/\sqrt{z_i^2 + 1}; \quad z_i = e^{-1/x_i} + x_i \cdot b.$$

Прийняти $x_i$ = 1,9; 1,5; 1,4; 1,2; $a_j$ = 0,2; 1,1; 1,4.

28. Обчислити значення:

$$y_{ij} = z_i^3 - (z_i + a_j \cdot b)/(a_j^{0,3} + z_i^2); \; z_i = e^{-2/x_i} + x_i \cdot b \cdot \cos(x_i).$$

Прийняти $x_i$ = 1,2; 1,6; 1,9; 2,2; $a_j$ = 0,75; 0,8; 0,9; 0,95; b = 6,74.

29. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в ***n*** різних точках ($x_i$, $y_i$), де ***n*** > 18; $x_{i+1} = x_i + e^{\delta x}$ ($x_0$ = 2; $\delta x$ = 0,1); $y_{i+1}$= dy*(1 – $y_i$) ($y_0$ =3; dy= -2).

$$F(x_i, y_i) = 15^{2x-3y} + \sin^2(x^3 + y^4) + x^2 y^3.$$

30. Обчислити значення:

$$y_{ij} = 0{,}88 \cdot z_i + \sqrt{\alpha \cdot z_i^2 + a_j^{2,5} \cdot z_i^{0,45}}; \qquad z_i = \lg(3{,}8 \cdot x_i).$$

Прийняти $x_i$ = 1,1; 1,4; 1,8; 2,2; 2,5; 3,1; $a_j$ = 0,2; 0,5; 0,9.

***Частина 2.***

1. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, а потім знайти суму всіх отриманих значень функції. Прийняти $n = 14$; $x_{i+1} = x_i + \delta x$ ($x_0 = 2{,}8$; $\delta x = 2{,}1$); $y_{i+1} = y_i + \delta y$ ($y_0 = 1{,}7$; $\delta y = 0{,}3$).

$$F(x_i, y_i) = \left[\sin\left(x_i^2\right) - \cos\left(y_i^2\right)\right] \frac{\sqrt[3]{2x_i - y_i}}{\cos(x_i)}.$$

2. Обчислити значення:

$$y_{ij} = z_i \cdot \cos^2 a_j + \sqrt{z_i^3 + a_j \cdot z_i}; \; z_i = \cos(x_i + 1).$$

Прийняти $x_i = 0{,}3; 0{,}8; 1{,}2$; $a_j = 1{,}1; 1{,}5; 1{,}9; 2{,}2$.

3. Обчислити значення:

$$y_{ij} = z_i^2 + \sqrt{z_i^2 - a_j}; \quad \text{где } z_i = e^{1/x_i} + x_i^2.$$

Прийняти $x_i = 1{,}2; 1{,}8; 1{,}6; 1{,}4$; $a_j = 0{,}4; 0{,}5; 0{,}8$.

4. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, де $n = 14$; $x_{i+1} = x_i + \delta x$ ($x_0 = 1?2$; $\delta x = 0?25$); $y_i = x_i^2$.

$$F(x_i, y_i) = \lg^2(x_i \cdot y_i) + \frac{\sqrt{x_i^2 + y_i^2}}{x_i + y_i}.$$

5. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, де $n = 15$; $x_{i+1} = x_i + \delta x$ ($x_0 = 4{,}2$; $\delta x = 1{,}8$); $y_{i+1} = y_i + \delta y$ ($y_0 = 1{,}82$; $\delta y = 0{,}33$).

$$F(x_i, y_i) = \left[\ln\left(x_i^2\right) - \ln\left(y_i^2\right)\right] \frac{\sqrt[3]{x_i - y_i}}{\operatorname{ctg}(x_i)};$$

6. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, де $n = 16$; $x_i$ та $y_i$ – довільні додатні числа.

$$F(x_i, y_i) = \sin^2(x_i^2 \cdot y_i) + \cos^3(x_i \cdot y_i^3) - \sqrt{\lg^3 x_i^2}.$$

7. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, де $n = 16$; $x_i$ – довільні додатні числа; $y_i = \lg^2 x_i^3 + (i-1)/10$.

$$F(x_i, y_i) = \operatorname{tg}^2(x_i - y_i) \cdot (x_i^2 + 0{,}5 \cdot y_i).$$

8. Обчислити значення:

$$y_{ij} = z_i + \sqrt{z_i^2 + a_j^3 \cdot z_i^{0,3}}; \qquad z_i = e^{-5 \cdot x_i}.$$

Прийняти $x_i = 0{,}1; 0{,}4; 0{,}8; 1{,}2$; $a_j = 0{,}2; 0{,}5; 0{,}9$.

9. Обчислити значення:

$$y_{ij} = (z_i^{\,2} + 0{,}33)/\operatorname{tg} z_i - b \cdot \sqrt{\left| z_i^{\,2} - a_j \right|};\ z_i = e^{-1/x_i} + x_i^{\,2}.$$

Прийняти $x_i$ = 1,3; 1,5; 1,7; 1,9; $a_j$ = 0,33; 1,12; 3,72; 5,16; b=3,68.

10. Обчислити значення:

$$y_{ij} = z_i^3 - (z_i + a_j \cdot b)/(a_j^{\,0,4} + z_i^2);\ z_i = e^{-1/x_i} + x_i \cdot b \cdot \cos x_i.$$

Потім знайти суму всіх отриманих значень функції $y_{ij}$.
Прийняти $x_i$ = 1,3; 1,5; 1,8; 2,0; $a_j$ = 0,7; 0,85; 0,9; b = 5,75.

11. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, а потім знайти добуток всіх отриманих значень функції.

$$F(x_i, y_i) = \lg^2(x_i \cdot y_i) + \frac{\sqrt{x_i^{\,2} + y_i^{\,2}}}{x_i + y_i}.$$

Прийняти *n* = 16; $x_{i+1} = x_i + \delta x$ ($x_0$ = 1,3; $\delta x$ = 0,2); $y_i = x_i^2$.

12. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$:

$$F(x_i, y_i) = e^{-1/x} + x^y - y^x,$$

де *n* = 15; $x_{i+1} = x_i + \delta x$ ($x_0$ = 1,2; $\delta x$ = 0,3); $y_i$ – довільні додатні числа.

13. Обчислити та запам'ятати значення:

$$y_{ij} = A_i^2 + A_i \cdot B_i \cdot C_j - B_i^2 \cdot \sqrt{A_i + B_i};\quad A_i = \sin^2 x_i;\quad B_i = e^{1/x_i} + 3,$$

а потім знайти суму всіх отриманих значень функції $y_{ij}$.
Прийняти $x_i$ = 1,8; 1,4; 1,1; 1,5; $C_j$ = 3,11; 2,21; 2,87.

14. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, а потім знайти суму перших п'яти значень та добуток двох останніх.

$$F(x_i, y_i) = e^{-2/x} + 0{,}8 \cdot x^y - 1{,}7 \cdot y^x.$$

Прийняти *n* = 18; $x_{i+1} = x_i + \delta x$ ($x_0$ = 1,2; $\delta x$=0,55); $y_i$ – довільні додатні числа.

15. Обчислити значення:

$$y_i = a_i^{b_i} + b_i^{a_i};\quad a_i = e^{-1/x_i} + x_i \cdot C;\quad b_i = \cos^2 x_i + x_i^{0,3}.$$

Прийняти $x_i$ = 0,5; 0,8; 0.9; 1,2; 1,3; 1.5; 1,7; 1,8; 1,9; 2.1; 2.3; 2.4; C=1,86.

16. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$,

$$F(x_i, y_i) = 12 \cdot x_i^{0.65} - (1{,}9 \cdot x_i^2 - 0{,}85 \cdot y_i^{0.8}) \cdot tg^2(x_i) / y_i.$$

де *n* = 18; $x_i$ – довільні додатні числа; $y_i$ – довільні від'ємні числа.

17. Обчислити та запам'ятати значення:

$$y_{ij} = (A_i \cdot \sin B_j + B_j \cdot \cos A_i) \cdot (A_i^{0,5} + \sqrt[3]{B_j});$$

$$A_i = \cos^2 x_i + x_i^3;\ B_j = \sin^2 a_j + a_j^{0,5}.$$

Прийняти $x_i$ = 1,2; 0,4; 1,3; 0,8; $a_j$ = 0,6; 0,9; 1,9.

18. Обчислити значення:

$$y_{ij} = z_i + (a_j \cdot z_i \cdot b + b^{0,3})/\sqrt{z_i^{\,2} + 1}; \quad z_i = e^{-1/x_i} + x_i \cdot b.$$

Прийняти $x_i$ = 1,2; 1,6; 1,9; 1,95; $a_j$ = 0,9; 0,95; 0,99; b=1,68.

19. Обчислити значення:

$$y_{ij} = z^{0,3} + z_i \cdot a_j; \qquad z_i = x_i^2 + \ln x_i.$$

Прийняти $x_i$ = 1,9; 1,5; 1,4; 1,2; $a_j$ = 0,2; 1,1; 1,4.

20. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках ($x_i$, $y_i$), а потім знайти суму перших п'яти значень та добуток двох останніх.

$$F(x_i, y_i) = 0{,}8 \cdot x^y - 1{,}7 \cdot y^x + \ln\frac{x}{y}.$$

Прийняти *n* = 16; $x_{i+1} = x_i + \delta x$ ($x_0$ = 1.5; $\delta x$=0,65); $y_i$ – довільні додатні числа.

21. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках ($x_i$, $y_i$),

$$F(x_i, y_i) = \frac{\sqrt{1 - y_i}}{x_i + \operatorname{tg} x_i} \cdot \left(1 + e^{-x_i \cdot y_i}\right).$$

де *n* = 16; $x = x_i * \delta x$ (x = 1,8; $\delta x$ = 1,2); $y_i$ – довільні додатні числа ($y_i$ < 1).

22. Обчислити значення:

$$y_{ij} = z_i^3 - (z_i + a_j \cdot b)/\left(a_j^{\,0,3} + z_i^2\right); \; z_i = e^{-2/x_i} + x_i \cdot b \cdot \cos(x_i).$$

Прийняти $x_i$ = 1,2; 1,6; 1,9; 2,2; $a_j$ = 0,75; 0,8; 0,9; 0,95; b = 6,74.

23. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках ($x_i$, $y_i$),

$$F(x_i, y_i) = 15^{2x-3y} + \sin^2(x^3 + y^4) + x^2 y^3,$$

де *n* = 18; $x_{i+1} = x_i + e^{\delta x}$ ($x_0$ = 2; $\delta x$ = 0,1); $y_{i+1}$= dy*(1 – $y_i$) ($y_0$ =3; dy= -2).

24. Обчислити значення:

$$y_{ij} = 0{,}88 \cdot z_i + \sqrt{\alpha \cdot z_i^2 + a_j^{2,5} \cdot z_i^{0,45}}; \qquad z_i = \lg(3{,}8 \cdot x_i).$$

Прийняти $x_i$ = 1,1; 1,4; 1,8; 2,2; 2,5; 3,1; $a_j$ = 0,2; 0,5; 0,9.

25. Обчислити та запам'ятати значення:

$$y_{ij} = A_i^{2,2} + 1{,}5 \cdot A_i \cdot B_i \cdot C_j - B_i^{1,5} \cdot \sqrt{A_i + 2B_i}; \qquad A_i = 1{,}8 \cdot \cos^3 x_i; \qquad B_i = e^{2/x_i} + 2{,}6,$$

а потім знайти суму всіх отриманих значень функції $y_{ij}$.
Прийняти $x_i$ = 2,2; 1,9; 1,7; 1,4; 1,2; $C_j$ = 2,31; 2,86; 3,57.

26. Обчислити значення:

$$y_i = a_i^{2b_i} + b_i^{3a_i}; \qquad a_i = e^{2,5/x_i} + 3x_i \cdot C; \qquad b_i = \sin^2 x_i + x_i^{0,5}.$$

Прийняти $x_i$ = 0,3; 0,7; 0.9; 1,1; 1,2; 1.4; 1,7; 1,8; 1,9; 2.2; 2.5; 2.7; C=2,15.

27. Обчислити значення:

$$y_{ij} = z_i \cdot \cos^2 a_j + \sqrt{z_i^3 + a_j \cdot z_i}; \; z_i = \cos(x_i + 1).$$

Прийняти $x_i$ = 0,3; 0,8; 1,2; $a_j$ = 1,1; 1,5; 1,9; 2,2.

28. Обчислити значення:

$$y_{ij} = z_i^{\,2} + \sqrt{z_i^{\,2} - a_j}; \quad \text{де } z_i = e^{1/x_i} + x_i^{\,2}.$$

Прийняти $x_i$ = 1,2; 1,8; 1,6; 1,4; $a_j$ = 0,4; 0,5; 0,8.

29. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, де *n* >14; $x_{i+1} = x_i + \delta x$ ($x_0$ = 1,2; $\delta x$ = 0,25); $y_i = x_i^2$.

$$F(x_i, y_i) = \lg^2(x_i \cdot y_i) + \frac{\sqrt{x_i^{\,2} + y_i^{\,2}}}{x_i + y_i}.$$

30. Обчислити та запам'ятати значення $z_i = F(x_i, y_i)$ в *n* різних точках $(x_i, y_i)$, де *n* > 20; $x_{i+1} = x_i + \delta x$ ($x_0$ = 3,2; $\delta x$ = 1,2); $y_{i+1} = y_i + \delta y$ ($y_0$ = 1,32; $\delta y$ = 0,33).

$$F(x_i, y_i) = \left[\ln\left(x_i^{\,2} - 1\right) - \ln\left(y_i^{\,2} + 1\right)\right] \frac{\sqrt[3]{x_i - 3y_i}}{\operatorname{tg}(x_i + 1)}.$$

## Завдання до комп'ютерного практикуму №5

***Частина 1.***

1. В довільному реченні замінити всі літери «а» на літеру «о» і надрукувати нове речення.
2. Одержати та надрукувати слово, яке утворено з перших букв слів довільного речення.
3. У кожному слові довільного речення замінити перші літери на великі і надрукувати нове речення.
4. Серед слів довільного речення вивести на екран тільки ті слова, що не мають в своєму складі літера «а».
5. У кожному слові довільного речення виділити та надрукувати перші три літери, якщо кількість літер у ньому парна, і перші дві літери – якщо непарна.
6. Організувати введення речення зі знаками пунктуації та вивести речення без знаків пунктуації.
7. Серед слів довільного речення знайти ті, у котрих однакові друга та третя літери.
8. У кожному слові довільного речення поміняти місцями першу та останню літери і надрукувати нове речення.
9. Привести (зменшити) всі слова довільного речення до однакової довжини (довжини найкоротшого слова).
10. Одержати та надрукувати слово, яке утворено з останніх літер слів довільного речення.
11. Організувати зчитування речення в рядкову змінну та замінити пробіли комою і вивести результуючий рядок.
12. Визначити у введеному реченні слова, у які не входить літера «к» та надрукувати їх.
13. Ввести довільне речення, яке містить цілі цифри. Замінити знайдені в реченні цифри знаком оклику.
14. Ввести в рядкову змінну довільне речення. Замінити всі малі літери на великі і вивести результуюче речення.
15. У кожному слові довільного речення поміняти місцями другу та передостанню букви і надрукувати нове речення.
16. Серед слів довільного речення визначити та надрукувати слова, які не починаються літерою «с».
17. Серед слів довільного речення знайти ті, у котрих однакові перша та остання літери.
18. Одержати та надрукувати слово, яке утворено з других літер слів довільного речення.
19. Організуйте введення довільного речення зі знаками пунктуації. Виведіть результуюче речення, видаливши з нього знаки пунктуації.
20. Організуйте введення довільної кількості слів. В кожному слові поміняйте першу та останню літеру місцями. Виведіть отримані слова.
21. В кожному слові довільного речення замініть останні літери на великі та виведіть отримане речення.
22. Організувати введення довільного речення. Першу літеру кожного слова замінити на порядковий номер слова у реченні.
23. Серед слів довільного речення визначити слова, які мають парну кількість букв, і дописати на початку кожного з цих слів кількість символів, з яких слово складається. Надрукувати речення до та після зміни.
24. Одержати та надрукувати слово, яке утворено з перших букв слів довільного речення та слово, яке утворене з останніх літер .

25. У кожному слові довільного речення замінити літеру «о» на літеру «а», підрахувати загальну кількість замін і надрукувати нове речення.
26. Серед слів довільного речення вивести на екран тільки ті слова, що не мають в своєму складі літери «а».
27. У кожному слові довільного речення виділити та надрукувати перші три літери, якщо кількість літер у ньому парна, і перші дві літери – якщо непарна.
28. Серед слів довільного речення обрати та вивести на екран тільки ті, що починаються або закінчуються на літеру «а».
29. Серед слів довільного речення знайти ті, у котрих однакові перша та третя літери.
30. У кожному слові довільного речення поміняти місцями першу та останню літеру, якщо в слові більше 2 літер, і надрукувати нове речення.

***Частина 2.***

1. В довільному масиві ***Z*** з ***n*** елементів знайти найменший додатний елемент та поміняти його місцями з першим елементом.
2. В довільному масиві ***T*** з ***n*** елементів знайти найбільший елемент кратний трьом та поміняти його місцями з останнім елементом.
3. В довільному масиві ***S*** з ***n*** елементів знайти суму та кількість від'ємних елементів, кратних чотирьом.
4. В довільному масиві ***E*** з ***n*** елементів знайти суму та кількість елементів, значення яких не менше заданої константи ( $E_i \geq \alpha$).
5. В довільному масиві ***V*** з ***n*** елементів знайти найменший елемент кратний п'яти та поміняти його місцями з першим елементом.
6. В довільному масиві ***Z*** з ***n*** елементів знайти найбільший елемент кратний чотирьом та поміняти його місцями з останнім елементом.
7. В довільному масиві ***Q*** з ***n*** елементів знайти найбільший від'ємний елемент та поміняти його місцями з першим елементом.
8. В довільному масиві ***G*** з ***n*** елементів знайти суму та кількість додатних елементів, кратних п'яти.
9. В довільному масиві ***B*** з ***n*** елементів знайти середнє значення $B_{cp}$, надрукувати елементи, значення яких більше середнього ($B_i > B_{cp}$) та знайти їх суму і кількість.
10. В довільному масиві ***B*** з ***n*** елементів знайти найменший елемент кратний двом та поміняти його місцями з першим елементом.
11. В довільному масиві ***P*** з ***n*** елементів знайти найбільший від'ємний елемент та поміняти його місцями з останнім елементом.
12. В довільному масиві ***W*** з ***n*** елементів знайти суму та кількість від'ємних елементів, кратних трьом.
13. В довільному масиві ***H*** з ***n*** елементів знайти найменший елемент кратний трьом та поміняти його місцями з першим елементом.
14. В довільному масиві ***U*** з ***n*** елементів знайти суму та кількість від'ємних елементів, кратних п'яти.
15. В довільному масиві ***L*** з ***n*** елементів знайти суму та кількість додатних елементів, кратних трьом.
16. В довільному масиві ***R*** з ***n*** елементів знайти найбільший елемент кратний двом та поміняти його місцями з останнім елементом
17. В довільному масиві ***R*** з ***n*** елементів знайти найменший елемент кратний чотирьом та поміняти його місцями з першим елементом.
18. В довільному масиві ***U*** з ***n*** елементів знайти найбільший та найменший елементи та поміняти їх місцями.
19. В довільному масиві ***B*** з ***n*** елементів знайти суму та кількість від'ємних елементів, кратних двом.
20. В довільному масиві ***Y*** з ***n*** елементів знайти найменший елемент кратний п'яти та поміняти його місцями з останнім елементом.
21. В довільному масиві ***S*** з ***n*** елементів знайти найбільший елемент кратний двом та поміняти його місцями з першим елементом.
22. В довільному масиві ***Q*** з ***n*** елементів знайти суму та кількість елементів, значення яких знаходяться в межах заданого інтервалу: $\alpha \leq Q_i \leq \beta$.

23. В довільному масиві ***G*** з ***n*** елементів знайти найменший елемент та поміняти його місцями з останнім елементом.
24. В довільному масиві ***X*** з ***n*** елементів знайти найменший елемент кратний чотирьом та поміняти його місцями з останнім елементом.
25. В довільному масиві ***F*** з ***n*** елементів знайти суму та кількість елементів, значення яких не перевищують заданої константи: $F_i \leq \alpha$.
26. В довільному масиві ***K*** з ***n*** елементів знайти найбільший елемент кратний трьом та поміняти його місцями з першим елементом.
27. В довільному масиві ***U*** з ***n*** елементів знайти найбільший елемент кратний п’яти та поміняти його місцями з першим елементом.
28. В довільному масиві ***G*** з ***n*** цілих чисел знайти суму, кількість та середнє значення парних елементів.
29. В довільному масиві ***Q*** з ***n*** елементів знайти найбільший від’ємний елемент та поміняти його місцями з найменшим додатнім елементом.
30. В довільному масиві ***H*** з ***n*** елементів знайти найменший елемент кратний трьом та поміняти його місцями з найбільшим елементом кратним трьом.

## Завдання до комп'ютерного практикуму №6

***Частина 1.***

1. В довільному масиві ***B*** з ***n*** дійсних елементів знайти середнє значення $B_{cp}$, надрукувати елементи, значення яких більше середнього ($B_i > B_{cp}$) та знайти їх суму, кількість та середнє значення.
2. В довільному масиві ***G*** з ***n*** дійсних елементів знайти суму та кількість додатних елементів, ціла частина яких кратна п'яти.
3. В довільному масиві ***Q*** з ***n*** дійсних додатніх та від'ємних елементів знайти найбільший від'ємний елемент та поміняти його місцями з першим додатнім елементом.
4. В довільному масиві ***Z*** з ***n*** елементів знайти найбільший елемент кратний чотирьом та поміняти його місцями з останнім елементом кратним двом.
5. В довільному масиві ***V*** з ***n*** дійсних елементів знайти найменший елемент, ціла частина якого кратна п'яти, та поміняти його місцями з першим елементом.
6. В довільному масиві ***E*** з ***n*** дійсних елементів знайти суму та кількість елементів, значення яких не менше заданої константи ( $E_i \geq \alpha$).
7. В довільному масиві ***S*** з ***n*** дійсних елементів знайти суму та кількість від'ємних елементів, ціла частина яких кратна трьом.
8. В довільному масиві ***U*** з ***n*** елементів знайти найбільший елемент кратний п'яти та поміняти його місцями з першим від'ємним елементом.
9. В довільному масиві ***K*** з ***n*** елементів знайти найбільший елемент кратний трьом та поміняти його місцями з першим не кратним трьом елементом.
10. В довільному масиві ***F*** з ***n*** дійсних додатніх та від'ємних елементів знайти суму та кількість елементів, значення яких за модулем не перевищують заданої константи: $F_i \leq \alpha$.
11. В довільному масиві ***X*** з ***n*** додатніх та від'ємних елементів знайти найменший елемент кратний чотирьом та поміняти його місцями з останнім від'ємним елементом.
12. В довільному масиві ***G*** з ***n*** дійсних додатніх та від'ємних елементів знайти найменший за модулем елемент та поміняти його місцями з останнім додатнім елементом.
13. В довільному масиві ***X*** з ***n*** дійсних елементів знайти суму та кількість від'ємних елементів, а також суму та кількість елементів, значення яких більше ніж середнє значення елементів масиву.
14. В довільному масиві ***Q*** з ***n*** дійсних додатніх та від'ємних елементів знайти суму та кількість елементів, значення за модулем яких знаходяться в межах заданого інтервалу: $\alpha \leq Q_i \leq \beta$.
15. В довільному масиві ***G*** з ***n*** дійсних від'ємних та додатніх елементів знайти найменший елемент та поміняти його місцями з останнім додатнім елементом масиву.
16. В довільному масиві ***S*** з ***n*** елементів знайти найбільший елемент кратний двом та поміняти його місцями з найменшим елементом.
17. В довільному масиві ***Y*** з ***n*** додатніх та від'ємних елементів знайти найменший елемент кратний п'яти та поміняти його місцями з останнім від'ємним елементом.
18. В довільному масиві ***T*** з ***n*** елементів знайти найбільший елемент кратний трьом та поміняти його місцями з останнім елементом кратним двом.
19. В довільному масиві ***B*** з ***n*** елементів знайти суму та кількість від'ємних елементів, кратних двом, а також суму та кількість додатніх елементів, кратних двом.
20. В довільному масиві ***U*** з ***n*** елементів знайти найбільший від'ємний та найменший додатній (крім нуля) елементи та поміняти їх місцями.

21. В довільному масиві ***R*** з ***n*** додатніх та від'ємних елементів знайти найбільший від'ємний елемент кратний чотирьом та поміняти його місцями з першим від'ємним елементом.
22. В довільному масиві ***R*** з ***n*** елементів знайти найбільший елемент кратний двом та поміняти його місцями з останнім кратним трьом елементом
23. В довільному масиві ***T*** з ***n*** дійсних елементів знайти найбільший елемент, ціла частина якого кратна трьом та поміняти його місцями з останнім елементом.
24. В довільному масиві ***L*** з ***n*** дійсних елементів знайти суму та кількість додатніх елементів, цілі частини яких кратні двом.
25. В довільному масиві ***U*** з ***n*** елементів знайти суму та кількість елементів, кратних двом та меньших за середнє значення елементів масиву.
26. В довільному масиві ***H*** з ***n*** елементів знайти найменший елемент кратний трьом та поміняти його місцями з середнім елементом масиву, якщо він є, інакше - з елементом ***n***/2.
27. В довільному масиві ***W*** з ***n*** елементів знайти суму та кількість від'ємних елементів, кратних трьом, а також суму та кількість додатніх елементів.
28. В довільному масиві ***P*** з ***n*** дійсних додатніх та від'ємних елементів знайти найбільший від'ємний елемент та поміняти його місцями з останнім додатнім елементом.
29. В довільному масиві ***B*** з ***n*** елементів знайти найменший елемент кратний двом та поміняти його місцями з першим елементом кратним трьом.
30. В довільному масиві ***Z*** з ***n*** дійсних додатніх та від'ємних елементів знайти найменший додатній елемент та поміняти його місцями з першим від'ємним елементом.

***Частина 2.***

1. Задано довільну цілочисельну матрицю ***S*(*m, n*)**. Діапазон значень елементів матриці [-50, 50]. В 1, 4, 7 і т.д. стовпцях матриці (окремо) визначити мінімальні за модулем елементи та їх координати. Матрицю ***S***, значення та координати мінімальних елементів надрукувати.
2. Задано довільну цілочисельну матрицю ***U(n, n)***. Діапазон значень елементів матриці [-350, 250]. Для елементів матриці ***U***, розташованих на її головній діагоналі знайти всі додатні елементи кратні, трьом, їх суму, кількість та середнє значення. Матрицю ***U***, отримані суму, кількість та середнє значення надрукувати.
3. Задано довільну цілочисельну матрицю ***B(m, n)***. Діапазон значень елементів матриці [-200, 200]. В 1, 4, 7 і т.д стовпцях матриці (окремо) визначити всі додатні елементи кратні трьом, їх суми, кількості та середні значення. Матрицю ***B***, отримані суми, кількості та середні значення надрукувати.
4. Задано довільну цілочисельну матрицю ***E(m, n)***. Діапазон значень елементів матриці [-250, 50]. В кожному рядку матриці окремо знайти суми, кількості та середні значення від'ємних елементів кратних трьом. Матрицю, отримані суми, кількості та середні значення надрукувати.
5. Задано довільну цілочисельну матрицю ***F(m, n)***. Діапазон значень елементів матриці [-200, 300]. В 2, 5, 8 і т.д рядках матриці (окремо) знайти суми, кількості та середні значення від'ємних елементів кратних, трьом. Матрицю ***F***, отримані суми, кількості та середні значення надрукувати.
6. Задано довільну цілочисельну матрицю ***C(m, n)***. Діапазон значень елементів матриці [-50, 100]. В 2, 5, 8 і т.д стовпцях матриці (окремо) визначити максимальні по модулю елементи та їх координати. Матрицю, значення та координати максимальних елементів надрукувати.
7. Задано довільну цілочисельну матрицю ***D(n, n)***. Діапазон значень елементів матриці [-100, 50]. Для елементів матриці, розташованих на її головній діагоналі знайти максимальний елемент серед кратних двом, та його координати. Надрукувати матрицю, максимальний елемент та його координати.
8. Задано довільну цілочисельну матрицю ***G(m, n)***. Діапазон значень елементів матриці [-100, 200]. В кожному з парних стовпців матриці визначити суми, кількості та середні значення додатних елементів, кратних двом. МАтрицю, отримані суми, кількості та середні значення надрукувати.
9. Задано довільну цілочисельну матрицю ***K(m, n)***. Діапазон значень елементів матриці [-200, 100]. В 2, 5, 8 і т.д стовпцях матриці (окремо) знайти максимальні елементи, кратні двом та їх координати. Надрукувати матрицю, максимальні елементи та їх координати.
10. Задано довільну цілочисельну матрицю ***A(n, n)***. Діапазон значень елементів матриці [-50, 150]. Для елементів матриці, які знаходяться на її побічній діагоналі визначити суму, кількість та середнє значення всіх від'ємних елементів, кратних двом. Матрицю, отримані суму, кількість та середнє значення надрукувати.
11. Задано довільну цілочисельну матрицю ***B(n, n)***. Діапазон значень елементів матриці [-150, 50]. Для елементів матриці, які знаходяться на її побічній діагоналі визначити мінімальні елементи, кратні трьом та їх координати. Матрицю, значення та координати мінімальних елементів надрукувати.
12. Задано довільну цілочисельну матрицю ***Y(m, n)***. Діапазон значень елементів матриці [-250, 200]. В 2, 5, 8 і т.д. стовпцях матриці ***Y*** (окремо) визначити суми, кількості та середні

значення від'ємних елементів, кратних трьом. Матрицю ***Y***, отримані суми, кількості та середні значення надрукувати.

13. Задано довільну цілочисельну матрицю ***Q(m, n)***. Діапазон значень елементів матриці [-200, 400]. В кожному непарному стовпці матриці ***Q*** визначити суми, кількості та середні значення додатних елементів кратних, чотирьом. Матрицю ***Q***, отримані суми, кількості та середні значення надрукувати.
14. Задано довільну цілочисельну матрицю ***A(m, n)***. Діапазон значень елементів матриці [-100, 100]. В кожному непарному стовпці матриці ***A*** (окремо) визначити суми, кількості та середні значення від'ємних елементів кратних п'яти. Матрицю ***A***, отримані суми, кількості та середні значення надрукувати.
15. Задано довільну цілочисельну матрицю ***L(m, n)***. Діапазон значень елементів матриці [-400, 400]. В кожному парному стовпці матриці ***L*** (окремо) знайти значення та координати максимальних елементів, кратних трьом. Матрицю, значення та координати максимальних елементів надрукувати.
16. Задано довільну цілочисельну матрицю ***P(m, n)***. Діапазон значень елементів матриці [-100, 300]. В кожному стовпці матриці окремо визначити суми, кількості та середні значення додатних елементів кратних, п'яти. Матрицю ***P***, отримані суми, кількості та середні значення надрукувати.
17. Задано довільну цілочисельну матрицю ***S(m, n)***. Діапазон значень елементів матриці [-150, 150]. В 1, 4, 7 і т.д рядках матриці ***S*** (окремо) визначити мінімальні елементи, кратні чотирьом та їх координати. Матрицю ***S***, значення та координати мінімальних елементів надрукувати.
18. Задано довільну цілочисельну матрицю ***R(m, n)***. Діапазон значень елементів матриці [-300, 100]. В 1, 4, 7 і т.д рядках матриці (окремо) визначити суми, кількості та середні значення додатних елементів кратних, п'яти. Матрицю ***R***, отримані суми, кількості та середні значення надрукувати.
19. Задано довільну цілочисельну матрицю ***A(m, n)***. Діапазон значень елементів матриці [-300, 150]. В кожному непарному рядку матриці (окремо) знайти максимальні елементи, кратні чотирьом. Надрукувати матрицю ***A***, максимальні елементи та їх координати.
20. Задано довільну цілочисельну матрицю ***D(n, n)***. Діапазон значень елементів матриці [-200, 300]. Для елементів матриці ***D***, які знаходяться на її побічній діагоналі визначити мінімальний елемент, кратний п'яти, та його координати. Надрукувати матрицю ***D***, значення та координати мінімального елементу.
21. Задано довільну цілочисельну матрицю ***B(m, n)***. Діапазон значень елементів матриці [-150, 250]. В кожному стовпці матриці окремо визначити суми, кількості та середні значення від'ємних елементів кратних, трьом. Матрицю ***B***, отримані суми, кількості та середні значення надрукувати.
22. Задано довільну цілочисельну матрицю ***D(m, n)***. Діапазон значень елементів матриці [-250, 150]. В кожному непарному стовпці матриці (окремо) визначити максимальні елементи, кратні п'яти, та їх координати. Надрукувати матрицю ***D***, значення та координати максимальних елементів.
23. Задано довільну цілочисельну матрицю ***C(m, n)***. Діапазон значень елементів матриці [-200, 150]. В кожному рядку матриці окремо визначити мінімальні по модулю елементи та їх координати. Матрицю ***C***, значення та координати мінімальних елементів надрукувати.
24. Задано довільну цілочисельну матрицю ***L(m, n)***. Діапазон значень елементів матриці [-300, 200]. В кожному парному рядку матриці (окремо) визначити суми, кількості та середні

значення додатних елементів кратних, чотирьом. Матрицю ***L***, отримані суми, кількості та середні значення надрукувати.

25. Задано довільну цілочисельну матрицю ***H(m, n)***. Діапазон значень елементів матриці [-400, 200]. В 2, 5, 8 і т.д. стовпцях матриці ***H*** (окремо) визначити суми, кількості та середні значення додатних елементів кратних, двом. Матрицю ***H***, отримані суми, кількості та середні значення надрукувати.
26. Задано довільну цілочисельну матрицю ***B(m, n)***. Діапазон значень елементів матриці [-150, 300]. В кожному непарному стовпці матриці (окремо) визначити суми, кількості та середні значення всіх від'ємних елементів кратних, двом. Матрицю ***B***, отримані суми, кількості та середні значення надрукувати.
27. Задано довільну цілочисельну матрицю ***P(m, n)***. Діапазон значень елементів матриці [-200, 200]. В кожному парному стовпці матриці окремо визначити суми, кількості та середні значення додатних елементів кратних, п'яти. Матрицю ***P***, отримані суми, кількості та середні значення надрукувати.
28. Задано довільну матрицю дійсних чисел ***D(m, n)***. Діапазон значень елементів матриці [-150, 150]. В кожному непарному рядку матриці окремо визначити мінімальні за модулем елементи та їх координати. Матрицю ***D***, значення та координати мінімальних елементів надрукувати.
29. Задано довільну матрицю дійсних чисел ***F(m, n)***. Діапазон значень елементів матриці [-300, 300]. В 2, 5, 8 і т.д рядках матриці (окремо) знайти суми, кількості та середні значення від'ємних елементів. Матрицю ***F***, отримані суми, кількості та середні значення надрукувати.
30. Задано довільну матрицю дійсних чисел ***A(n, n)***. Діапазон значень елементів матриці [-100, 150]. Для елементів матриці, які знаходяться на її побічній діагоналі визначити суму, кількість та середнє значення всіх від'ємних елементів, ціла частина яких кратна двом. Матрицю, отримані суму, кількість та середнє значення надрукувати.

## Завдання до комп'ютерного практикуму №7

1. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних wordCount з двома полями: word та wordn типу string. Створити власну структуру даних count_234n з полями count2, count3, count4 та countn типу unsigned. Створити вектор vecCount типу vector<wordCount>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vecCount.word. Підрахувати окремо кількість слів в яких 2, 3 та 4 символи, а також всі інші слова та записати отримані значення у поля count_234n.count2, count_234n.count3, count_234n.count4 та count_234n.countn відповідно. В кінець кожного слова – елементу вектора, дотати цілочисельне значення, яке відповідає кількості символів у слові (наприклад, «слово5»). Записати слова після додавання в кінець кожного слова кількості літер у поле вектору vecCount.wordn. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до додавання кількості символів та після. Вивести кількість слів з 2-ма, 3-ма та 4-ма символами, а також всіх інших слів.
2. Задати довільне речення з клавіатури чи з зовнішнього файлу. Підрахувати кількість символів в кожному слові. Створити власну структуру даних vecStr, яка складатиметься з чотирьох векторів типу vector<string>. Слова з кількістю літер 3 записати у поле вектор vecStr.vec3, слова з кількістю літер 4 записати у вектор vecStr .vec4, слова з кількістю літер 5 записати у вектор vecStr.vec5, всі інші слова записати у вектор vecStr.vecn. Вивести сформовані вектори та кількість елементів в них і початкове речення на екран та (або) у зовнішній файл.
3. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних word_replc з трьома полями: word та word_aie типу string, та полем replc_count типу unsigned. Створити вектор vec_replc типу vector<word_replc>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vec_replc.word. Знайти у кожному слові літери «а», «і» та «е» та замінити їх на великі. Записати слова після замін у поле вектору vec_replc.word_aie. Підрахувати кількість замін у кожному слові та записати у поле vec_replc.replc_count. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер «а», «і» та «е» та після, а також кількість зроблених в кожному слові замін.
4. Задати довільне речення з клавіатури чи з зовнішнього файлу. Записати всі слова окрім пробілів і знаків пунктуації у вектор. Створити власну структуру даних з чотирма полями: для запису слова (типу string), кількості літер «k» (типу unsigned), кількості літер «і» (типу unsigned) та кількості літер «с» (типу unsigned). Створити вектор типу створеної структури даних. Записати у створений вектор кожне слово та кількість знайдених в ньому літер «k», «і» та «с». Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок та вказати знайдену в ньому кількість літер «k», «і» та «с».
5. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних char_123n з чотирма полями: word типу vector<string>, char_1, char_2, char_3 типу vector<char>та str_n типу vector<string>. Створити змінну vec_123n типу створеної структури даних char_123n. Записати всі слова окрім пробілів і знаків пунктуації у вектор vec_123n.word. Сформувати вектор vec_123n.char_1 з перших літер всіх слів, вектор vec_123n.char_2 з других літер всіх слів, вектор vec_123n.char_3 з третіх літер всіх слів та вектор vec_123n.str_n, елементами якого будуть залишки слів без перших трьох літер. Вивести на екран та (або) у зовнішній файл введене речення, а також всі створені вектори.

6. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних rplcmnt з двома полями word та wordrplc типу string. Створити вектор vecrplc типу створеної структури даних vector<rplcmnt>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vecrplc.word. Знайти у кожному слові літеру «j» та замінити її на «1», літеру «i» та замінити її на «2», літеру «k» та замінити її на «3», інші літери в слові змінити на великі. Записати кожне слово після заміни символів у поле вектору vecrplc.wordrplc. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер «j», «i» та «k» та після заміни їх на «1», «2» та «3».
7. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних struc_char з шістьма полями: word та str_n типу string і полямим char_count, char_1, char_n, char_n1 типу unsigned. Створити вектор vec_char типу vector<struc_char>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vec_char.word. Сформувати вектор vec_char.char_1 з перших літер всіх слів, вектор vec_char.char_n з останніх літер всіх слів, вектор vec_char.char_n1 з передостанніх літер всіх слів та вектор vec_char.str_n, елементами якого будуть залишки слів без першої та двох останніх літер. Записати в поле vec_char.char_count кількість символів в кожному введеному користувачем слові. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремих рядках поля створеного вектору - слово, кількість в ньому літер, залишок слова без першої та двох останніх літер, першу літеру слова, останню літеру слова та передостанню літеру слова.
8. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних replacmnt з двома полями word та word13n типу string. Створити вектор vecStrc типу створеної структури даних vector<replacmnt>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vecStrc.word. У кожному слові замінити першу, третю та останню літеру на велику, інші літери в слові змінити на «0». Записати слова після заміни у поле вектору vecStrc.word13n. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер та після заміни їх.
9. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних wrd_cnt з двома полями: wrd та n_wrd типу string. Створити власну структуру даних cnt_135n з чотирма полями: cnt_1, cnt_3, cnt_5 та cnt_n типу unsigned. Створити вектор vec_cnt типу vector<wrd_cnt>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vec_cnt.wrd. Створити змінну var_135n типу створеної структури даних cnt_135n. Підрахувати окремо кількість слів в яких 1, 3 та 5 символів, а також всі інші слова та записати отримані значення у поля var_135n.cnt_1, var_135n.cnt_3, var_135n.cnt_5 та var_135n.cnt_n відповідно. На початок кожного слова – елементу вектора, дотати цілочисельне значення, яке відповідає кількості символів у слові (наприклад, «5_слово») та записати його у поле вектору vec_cnt.n_wrd. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до додавання кількості символів та після. Вивести кількість слів з 1-им, 3-ма та 5-ма символами, а також всіх інших слів.
10. . Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних text_rty з полями word та bWord типу string та полями char_r, char_t, char_y, char_else типу unsigned. Створити вектор vect_rty типу vector<text_rty>. Підрахувати окремо кількість символів «r», «t», «y» та всіх інших у кожному слові і записати їх у поля вектору vect_rty.char_r, vect_rty.char_t, vect_rty.char_y та vect_rty.char_else відповідно. Замінити перші літери кожного слова на великі та записати слова після заміни у поле вектору vect_rty.bWord. Вивести на екран та (або) у зовнішній файл введене речення, а також в

окремих рядках кожне слово до, та після заміни літер з вказаною після нього підрахованою кількість літер «r», «t», «y» та всіх інших у кожному слові.

11. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних word_rplc з трьома полями: word та word_n типу string, та полем word_cnt типу unsigned. Створити вектор vec_rplc типу vector<word_rplc>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vec_rplc.word. Замінити в кожному слові 1-шу, 3-тю та 5-ту літери порядковим номером кожної заміни у всьому введеному реченні (наприклад, «Замінити певні літери у кожному слові» - «1а2і3ити», «4е5н6», «7і8е9и», «10», «11о12н13му», «14л15в16»). Записати слова після замін у поле вектору vec_rplc.word_n. Підрахувати кількість замін у кожному слові та записати у поле вектору vec_rplc.word_cnt. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни та після, та кількість замін в кожному слові. Вивести загальну кількість замін в усьому реченні.
12. Задати довільний текст з клавіатури чи з зовнішнього файлу, що складається з кількох речень зі знаками пунктуації. Створити власну структуру даних punct_struct, яка скрадатиметься з наступних полів: capitl типу vector<string>; coma_c, dot_c та punct_c типу string::size_type. Підрахувати кількість слів, які починаються з великої літери, та записати їх у поле вектор capitl. Підрахувати у тексті кількість ком, записати її у поле coma_c та замінити коми на крапку з комою. Підрахувати у тексті кількість крапок, записати її у поле dot_c та замінити крапки на двокрапку. Підрахувати інші знаки пунктуації та записати їх кількість у поле punct_c. Вивести на екран та (або) у зовнішній файл введений текст, текст після заміни ком і крапок, а також кількість ком, кількість крапок та кількість інших знаків пунктуації в тексті.
13. Задати довільне речення з клавіатури чи з зовнішнього файлу, яке складається зі слів записаних великими літерами. Створити структуру даних struct_cnt з наступними полями даних: word та word_sml типу string; a_cnt, o_cnt, e_cnt та n_cnt типу unsigned. Створити вектор vect_oae типу vector<struct_cnt>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vect_oae.word. Підрахувати кількість літер «A», «O», «E» та замініти їх на малі літери, а також окремо підрахувати кількість всіх інших літер в кожному слові та записати їх у поля вектору vect_oae.a_cnt, vect_oae.o_cnt, vect_oae.e_cnt і vect_oae.n_cnt відповідно. Слова після заміни літер записати у поле вектору vect_oae.word_sml. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремих рядках кожне слово до та після заміни літер з вказаною після нього підрахованою кількість літер «A», «O», «E» та всіх інших літер у кожному слові
14. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних repl_struc з наступними полями: wrd, wrd_13n та wrd_pair типу string; count_pair типу string::size_type. Створити вектор vect_repl типу vector<repl_struc>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vect_repl.wrd. Замінити першу, третю та останню літеру в кожному слові на велику і записати слова після замін у поле вектору vect_repl.wrd_13n та підрахувати загальну кількість таких замін у реченні. Всі парні літери кожного слова замінити на 0 і записати слова після замін у поле вектору vect_repl.wrd_ pair та підрахувати їх кількість в кожному слові і зберегти у полі vect_repl.count_pair. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер та після заміни їх, кількість парних літер у кожному слові та загальну кількість замін у реченні 1-их, 3-іх та останніх літер.

15. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних sentence_strc, яка міститиме поля: sentnc, sentnc_a, sentnc_o та sentnc_e типу string; count_a, count_o та count_e типу string::size_type; vec_wrd типу vector<string>. Зберегти копії речення у полях структури sentnc, sentnc_a, sentnc_o та sentnc_e. Записати всі слова окрім пробілів і знаків пунктуації у поле вектор vec_wrd. Підрахувати у реченні кількість літер «a», «o» та «e» та зберегти знайдені значення у поля count_a, count_o та count_e. Замінити кожну літеру «a» у полі sentnc_a цілочисельним значенням, що відповідає порядковому номеру літери «a» в реченні (наприклад, «Замінити літери ‘a’ порядковими значеннями їх у реченні» – «31мінити літери ‘2’ порядковими зн3ченнями їх у реченні»). Вивести на екран та (або) у зовнішній файл введене речення, а також речення після заміни літер «a». Аналогічно замінити літери «o» у полі sentnc_o та «e» у полі sentnc_e, та вивести на екран речення після замін. Вивести слова з яких складається речення за допомогою поля-вектору vec_wrd.
16. Задати довільний текст з клавіатури чи з зовнішнього файлу, який складається з кількох речень зі знаками пунктуації. Створити власну структуру даних struct_aoe, що складається з наступних полів: wrd, wrd_big та wrd_repl типу string; count_little, count_little_a, count_little_o та count_little_e типу string::size_type. Створити вектор replvec типу vector<struct_aoe>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору replvec.wrd. Замінити в кожному слові останні літери на великі та зберегти слова у поля вектору replvec.wrd_big. Підрахувати в кожному слові кількість малих літер та записати у поле вектору replvec.count_little. Замінити малі літери «a» на 1, малі літери «o» на 2, а малі літери «e» на 3 і зберегти слова після заміни у полях вектору replvec.wrd_repl. Підрахувати в кожному слові кількість літер «a», «o» і «e» та записати її у поля вектору replvec.count_little_a, replvec.count_little_o і replvec.count_little_e відповідно. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремому рядку слово до заміни літер, після заміни останніх літер та після заміни літер «a», «o» і «e» з вказуванням кількості малих літер в слові, та окремо малих літер «a», «o» і «e».
17. Задати довільне речення з клавіатури чи з зовнішнього файлу, яке складається зі слів записаних великими літерами. Створити власну структуру даних struct_aoi, що складається з наступних полів: wrd, wrd_repl типу string; acount, ocount, icount і ncount типу string::size_type. Створити вектор replvec типу vector<struct_aoi>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору replvec.wrd. Підрахувати кількість літер «A», «O» та «I» в кожному слові: записати підраховану кількість у поля replvec.acount, replvec.ocount та replvec.icount відповідно; замінити знайдені літери на малі літери і зберегти отримане слово у поле вектору replvec.wrd_repl. Окремо підрахувати кількість всіх інших літер у кожному слові і записати її у поле replvec.ncount. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремі рядки слово до та після заміни літер, кількість у слові літер «A», «O», «I» та всіх інших літер.
18. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних struct_char, що складається з наступних полів: wrd, wrd_char та wrd_big типу string; pos_l1, pos_l2 та pos_l3 типу string::size_type. Створити вектор replvec типу vector<struct_char>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору replvec.wrd. Ввести три різних літери l1, l2, l3, які необхідно знайти у словах. Знайти першу позицію, в якій зустрічається кожна з трьох літер у слові і записати позиції літер у поля replvec.pos_l1, replvec.pos_l2 і replvec.pos_l3 відповідно. Замінити першу знайдену літеру l1 на одиницю, першу знайдену літеру l2 на двійку та першу знайдену літеру l3 на трійку і

зберегти слова після змін у поле вектору replvec.wrd_char. Замінити в кожному слові першу, другу та останню літери на великі і зберегти слова після змін у поле вектору replvec.wrd_big. Вивести на екран та (або) у зовнішній файл введене речення, а також у окремих рядках вхідне слово, слово після заміни літер l1, l2, l3 та слово після заміни літер на великі. Вказати позицію в слові, в якій вперше зустрілась кожна з літер l1, l2, l3.

19. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних swops_strc, яка міститиме поля: swops, swops_i, swops_o та swops_e типу string; count_i, count_o та count_e типу string::size_type; vec_wrd типу vector<string>. Зберегти копії речення у полях структури swops, swops_i, swops_o та swops_e. Записати всі слова окрім пробілів і знаків пунктуації у поле вектор vec_wrd. Підрахувати у реченні кількість літер «і», «о» та «е» та зберегти знайдені значення у поля count_i, count_o та count_e. Замінити кожну літеру «і» у полі swops_i цілочисельним значенням, що відповідає порядковому номеру літери «і» в реченні (наприклад, «Замінити літери ‘і’ порядковими значеннями їх у реченні» – «Зам4нити л11тери ‘18’ порядковими значеннями їх у реченн55»). Вивести на екран та (або) у зовнішній файл введене речення, а також речення після заміни літер «і». Аналогічно замінити літери «о» у полі swops_o та «е» у полі swops_e, та вивести на екран речення після замін. Вивести слова з яких складається речення за допомогою поля-вектору vec_wrd.
20. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних repl_struc з наступними полями: wrd, wrd_13n та wrd_pair типу string; count_pair типу string::size_type. Створити вектор vect_repl типу vector<repl_struc>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vect_repl.wrd. Замінити другу, четверту та останню літеру в кожному слові на велику і записати слова після замін у поле вектору vect_repl.wrd_13n та підрахувати загальну кількість таких замін у реченні. Всі непарні літери кожного слова замінити на «1» і записати слова після замін у поле вектору vect_repl.wrd_ pair та підрахувати їх кількість в кожному слові і зберегти у полі vect_repl.count_pair. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер та після заміни їх, кількість непарних літер у кожному слові та загальну кількість замін у реченні 2-их, 4-іх та останніх літер.
21. Задати довільне речення з клавіатури чи з зовнішнього файлу, яке складається зі слів записаних малими літерами. Створити власну структуру даних struct_my, що складається з наступних полів: wrd, wrd_repl типу string; a_cnt, o_cnt, i_cnt і n_cnt типу string::size_type. Створити вектор chng типу vector<struct_my>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору chng.wrd. Підрахувати кількість літер «а», «о» та «і» в кожному слові: записати підраховану кількість у поля chng.a_cnt, chng.o_cnt та chng.i_cnt відповідно; замінити знайдені літери на великі літери і зберегти отримане слово у поле вектору chng.wrd_repl. Окремо підрахувати кількість всіх інших літер у кожному слові і записати її у поле chng.n_cnt. Вивести на екран та (або) у зовнішній файл введене речення, а також в окремі рядки слово до та після заміни літер, кількість у слові літер «а», «о», «і» та всіх інших літер.
22. Задати довільне речення з клавіатури чи з зовнішнього файлу. Підрахувати кількість символів в кожному слові, загальну кількість символів, та кількість літер (без пробілів та знаків пунктуації). Створити власну структуру даних vecStr, яка складатиметься з чотирьох векторів типу vector<string>. Слова з кількістю літер від 1 до 3 записати у поле вектор vecStr.vec3, слова з кількістю літер від 4 до 6 записати у вектор vecStr .vec4, слова з кількістю літер від 7 до 9 записати у вектор vecStr.vec7, всі інші слова записати у вектор vecStr.vecn.

Вивести сформовані вектори та кількість елементів в них і початкове речення на екран та (або) у зовнішній файл.

23. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних repl_struc з наступними полями: wrd, wrd_13n та wrd_pair типу string; count_pair типу string::size_type. Створити вектор vect_repl типу vector<repl_struc>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vect_repl.wrd. Замінити першу, третю та останню літеру в кожному слові на порядковий номер літері у слові і записати слова після замін у поле вектору vect_repl.wrd_13n та підрахувати загальну кількість таких замін у реченні. Всі парні літери кожного слова замінити на великі і записати слова після замін у поле вектору vect_repl.wrd_ pair та підрахувати їх кількість в кожному слові і зберегти у полі vect_repl.count_pair. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер та після заміни їх, кількість парних літер у кожному слові та загальну кількість замін у реченні 1-их, 3-іх та останніх літер.
24. Задати довільний текст з клавіатури чи з зовнішнього файлу, що складається з кількох речень зі знаками пунктуації. Створити власну структуру даних pnct_strct, яка скрадатиметься з наступних полів: cptl типу vector<string>; com_c, dot_c та pnct_c типу string::size_type. Підрахувати кількість слів, які починаються з великої літери, замінити перші літери на малі та записати їх у поле вектор cptl. Підрахувати у тексті кількість ком, записати її у поле com_c та замінити коми на крапку з комою. Підрахувати у тексті кількість крапок, записати її у поле dot_c та замінити крапки на двокрапку. Підрахувати інші знаки пунктуації та записати їх кількість у поле pnct_c. Вивести на екран та (або) у зовнішній файл введений текст, текст після заміни ком і крапок, а також кількість ком, кількість крапок та кількість інших знаків пунктуації в тексті.
25. Задати довільне речення з клавіатури чи з зовнішнього файлу. Створити власну структуру даних rplcmnt з двома полями word та wordrplc типу string. Створити вектор vecrplc типу створеної структури даних vector<rplcmnt>. Записати всі слова окрім пробілів і знаків пунктуації у поле вектору vecrplc.word. Знайти у кожному слові літеру «l», літеру «o», літеру «k» та замінити їх на великі, інші літери в слові змінити на малі. Записати кожне слово після заміни символів у поле вектору vecrplc.wordrplc. Вивести на екран та (або) у зовнішній файл введене речення, а також кожне слово в окремий рядок до заміни літер «l», «o» та «k» та після заміни їх.

## Завдання до комп'ютерного практикуму №8

Розробити програму (відповідно своєму варіанту завдання), яка реалізує поставлену задачу у зовнішній функції (функціях) користувача. Зберігати результати розрахунку та вхідні дані за допомогою власної структури даних. Розрахунки на кожній ітерації зберігати у вектор. Створити зовнішню власну функцію для виведення елементів вектору на екран та у зовнішній текстовий файл. Передбачити в програмі обробку виключних ситуацій.

При організації функції передбачити друк доданків (співмножників) та поточної суми (добутку) на кожному кроці в залежності від побажання користувача.

Якщо знаменник доданка (співмножника) дорівнює нулю, такий доданок (співмножник) не враховувати та надрукувати відповідне повідомлення зі значенням аргументу.

Налаштувати програму та перевірити її роботу. Продемонструвати роботу програми викладачу.

| Варіант | Добуток, який треба розрахувати | Варіант | Сума, яку треба розрахувати |
|---|---|---|---|
| 1 | $P=\prod_{j=m}^{K}\frac{j^2-3j+1}{(j-4)(j-2)}\cdot(a-3)$ | 2 | $S=\sum_{n=J}^{K}\frac{(n+3)^3}{(n-5)^2}\cdot(a+1)$ |
| 3 | $P=\prod_{n=z}^{K}\frac{n+2}{(n-\alpha)(n-\beta)}\cdot(a+2)$ | 4 | $S=\sum_{k=L}^{M}\frac{3k^2}{4(k+1)(k-2)}\cdot(a-4)$ |
| 5 | $P=\prod_{i=L}^{m}\frac{2i+1}{(i-3\}(i+1)}\cdot(a+3)$ | 6 | $S=\sum_{n=K}^{M}\frac{n+11}{(n-6)(n-5)}\cdot(a-1)$ |
| 7 | $P=\prod_{l=i}^{j}\frac{l^2}{(l-3)(l-\varphi)}\cdot(a-5)$ | 8 | $S=\sum_{i=K1}^{N}\frac{i+2}{(i-3)(i-5)}\cdot(a+4)$ |
| 9 | $P=\prod_{k=m}^{n}\frac{k+8}{(k^2-9k+21)\cdot(a-4)}$ | 10 | $S=\sum_{n=K}^{I}\frac{(n+\beta)^2}{(n-4)(n-8)}\cdot(a-3)$ |
| 11 | $P=\prod_{l=M}^{N}\frac{l^2+3l}{(l-4)(l+5)}\cdot(a+4)$ | 12 | $S=\sum_{i=K}^{M}\frac{(i+1)^3}{(i^2-i-7)\cdot(a-5)}$ |
| 13 | $P=\prod_{i=M}^{N}\frac{i^2+2i+3}{i-3}\cdot(a-2)$ | 14 | $S=\sum_{n=i}^{K}\frac{n}{n^2-5n+6}\cdot(a-4)$ |
| 15 | $P=\prod_{k=i}^{j}\frac{k^2+4k+7}{(k-5)}\cdot(a+1)$ | 16 | $S=\sum_{j=Z1}^{Z2}\frac{(j+2)^3}{(j-8)(j-k)}\cdot(a+3)$ |
| 17 | $P=\prod_{k=M}^{N}\frac{k+3}{(k-5)(k-6)}\cdot(a-3)$ | 18 | $S=\sum_{n=L}^{K}\frac{n-2}{n^2-16}\cdot(a+5)$ |
| 19 | $P=\prod_{k=M}^{L}\frac{k+1}{(k-8)(k-6)}\cdot(a-1)$ | 20 | $S=\sum_{i=K}^{M}\frac{(i+1)^2}{(2i^3-3i-11)\cdot(a-4)}$ |
| 21 | $P=\prod_{n=z}^{K}\frac{n}{(n-2)(n-5)}\cdot(a+2)$ | 22 | $S=\sum_{n=K}^{L}\frac{(n+4)^3}{(n^3-n^2+2n-3)\cdot(a-3)}$ |
| 23 | $P=\prod_{k=N}^{M}\frac{k+1}{(k-5)(k-7)}\cdot(a-2)$ | 24 | $S=\sum_{n=L}^{k}\frac{n^2-n}{(n^2+n+6)\cdot(a+4)}$ |

| | | | |
|---|---|---|---|
| **25** | $P = \prod_{n=M}^{K} \frac{n}{(n-3)(n-8)} \cdot (a+5)$ | **26** | $S = \sum_{i=M}^{k} \frac{i^2 - 4i + 7}{(i^2 - 6i + 13) \cdot (a+2)}$ |
| **27** | $P = \prod_{n=L}^{m} \frac{(n+3)^3}{(n-5)^2} \cdot (a-3)$ | **28** | $S = \sum_{i=K}^{N} \frac{i^2 - i + 3}{i-5} \cdot (a+1)$ |
| **29** | $P = \prod_{l=i}^{j} \frac{(l+3)^2}{(l-7)(l-9)} \cdot (a-4)$ | **30** | $S = \sum_{i=K}^{J} \frac{i(i+1)}{(i+5)(i-3)} \cdot (a-2)$ |

## Завдання до комп'ютерного практикуму №9

Обчислити інтеграл для парних варіантів за формулою трапецій, а для непарних варіантів за формулою Сімпсона з точністю до 0,001. Величину кроку h, що забезпечує задану точність, визначити за допомогою подвійного перерахунку.

1. $\int_a^b \frac{dx}{\sqrt{2x^2+3,4}}$
2. $\int_a^b e^{-3x^2+x+0,5}dx$
3. $\int_a^b \frac{dx}{\sqrt{3x^3+2}}$
4. $\int_a^b \frac{\ln(3+x^2)}{3+2,5x^2}dx$
5. $\int_a^b \sqrt{\frac{1-0,35x^2}{1-0,5x^2}}\,dx$
6. $\int_a^b e^{-5x^2+3x+1}dx$
7. $\int_a^b \sqrt{\frac{2-0,3x}{3-2x^2}}dx$
8. $\int_a^b \sqrt{1,3+4,2x^3}dx$
9. $\int_a^b \frac{\ln(2+x)}{1,2x}dx$
10. $\int_a^b \frac{\ln(1,5+\sqrt{x})}{\sqrt[3]{\sqrt{x}+1}}dx$
11. $\int_a^b \frac{\ln(0,3x+2,5)}{\sqrt{x^2+0,8}}dx$
12. $\int_a^b \sqrt{x+1}\lg(x+3,4)dx$
13. $\int_a^b \frac{xdx}{\sqrt{1,4x^2+0,7}}$
14. $\int_a^b \frac{\cos x}{x+2}dx$
15. $\int_a^b \frac{dx}{\sqrt{x^2-0,9}}$
16. $\int_a^b \sqrt{2x}\cos x^2 dx$
17. $\int_a^b \frac{\ln(x^2+0,9)}{x-1}dx$
18. $\int_a^b \frac{dx}{\sqrt{2x^2+0,5x}}$
19. $\int_a^b \sqrt{\frac{2x+0,5}{1+0,2x^2}}dx$
20. $\int_a^b (1,8x+0,7)\sin x dx$
21. $\int_a^b \ln(1+x^2)dx$
22. $\int_a^b e^{-0,5x^2}\sin(0,7x)dx$
23. $\int_a^b \frac{\sqrt{1,1+x^2}}{1+\cos(1,1x)}dx$
24. $\int_a^b \frac{e^{-x^2}\sin(0,5x)}{0,6+x^2}\,dx$
25. $\int_a^b \frac{\ln(x^2+2)}{2x+1}dx$
26. $\int_a^b \frac{dx}{\sqrt{(1-x^2)(1-0,35x^2)}}$
27. $\int_a^b \frac{\ln(3,5+\sqrt[3]{x^2})}{\sqrt[3]{x+3x}}dx$
28. $\int_a^b \frac{\sqrt{1,3+4,2x^3}}{3(x^2-2x)}dx$
29. $\int_a^b \frac{e^{3x^2+2}}{3x+1}dx$
30. $\int_a^b \frac{\sqrt{x^3+1}}{\lg(x+3,4)}\,dx$

## Формули для довідок

1. Рівняння Менделєєва-Клапейрона:

$$P \cdot V = \frac{m}{M} \cdot R \cdot T,$$

де *P* – тиск; *V* – об'єм; *m* – маса газу; *M* – молекулярна маса; *R* – універсальна газова стала; *T* – абсолютна температура.

2. Співвідношення між елементами трикутника зі сторонами a, b, c та протилежними кутами A, B, C.

2.1. Теорема синусів:

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

де R - радіус описаного кола.

2.2. Теорема косинусів:

$$a^2 = b^2 + c^2 - 2bc \cos A.$$

2.3. Площа трикутника:

$$S = \frac{a\, h_a}{2}$$

де $h_a$ висота трикутника.

2.4. Форма Герона:

$$S = \sqrt{p\,(p-a)(p-b)(p-c)}, \qquad \left(p = \frac{a+b+c}{2}\right).$$

2.5. $$S = \frac{1}{2}\, a\, b \sin c.$$

2.6. $$S = r \cdot p,$$

$r -$ радіус кола,вписаного у трикутнок.

2.7. Для рівнобічного трикутника:

$$S = \frac{a^2 \sqrt{3}}{4}.$$

3. Площа фігур.

3.1. Паралелограма:

$$S = b\, h,$$

де $b$ – сторона, $h$ - висота.

3.2. Ромба:

$$S = \frac{1}{2}\, d_1\, d_2$$

де $d_1$ та $d_2$ – діагоналі.

3.3. Трапеції:

$$S = \frac{a+b}{2}\, h,$$

де $a$ та $b$ – сторони, $h$ - висота.

$$S = m\, h,$$

де - $m$ – середня лінія.

3.4. Квадрата:

$$S = a^2,$$

де $a$ – сторона.

3.5. Прямокутника:

$$S = a\, b,$$

де $a$ та $b$ – сторони.

3.6. Кола:

$$S = \pi\, R^2,$$

де $R$ – радіус.

4. Об'єми різних фігур:
де S – площа основи, H –висота.

4.1. Призми:

$$V = S\, H.$$

4.2. Циліндра:

$$V = S\, H.$$

4.3. Піраміди:

$$V = \frac{1}{3}\acute{S} \cdot H$$

4.4. Конуса:

$$V = \tfrac{1}{3} \cdot \acute{S} \cdot H,$$

де

$$\acute{S} = \pi\, R^2$$

4.5 Кулі:

$$V = \frac{4}{3}\, \pi\, R^3$$

де $R$ – радіус.

5. Квадратні рівняння:

5.1. $a\, x^2 + b\, x + c = 0;$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4\, a\, c}}{2\, a}.$$

5.2. $x^2 + p\, x + q = 0;$

$$x_{1,2} = -\frac{p}{2} + \sqrt{\frac{p^2}{4} - q}.$$

5.3. Теорема Віета:

коли $x^2 + p \cdot x + q = 0$, то $x_1 + x_2 = -p$, $x_1 \cdot x_2 = q$.

6. Рішення системи рівнянь з двома невідомими:

Для $\begin{cases} a_1 \cdot x + b_1 \cdot y = c_1, \\ a_2 \cdot x + b_2 \cdot y = c_2, \end{cases}$

$\Delta = a_1 \cdot b_2 - a_2 \cdot b_1$;
$\Delta_x = c_1 \cdot b_2 - c_2 \cdot b_1$;
$\Delta_y = a_1 \cdot c_2 - a_2 \cdot c_1$.

Тоді $x = \frac{\Delta_x}{\Delta}$; $y = \frac{\Delta_y}{\Delta}$

7. Довжина відрізка АВ ($A(x_1,\ y_1), B(x_2,\ y_2)$),

$$l_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

8. Яущо точка $D(x,\ y)$ знаходиться у площі трикутника АВС, де $A(x_1,\ y_1), B(x_2,\ y_2), C(x_3,\ y_3)$, то

$$S_{ABC} = S_{ADC} + S_{CDB} + S_{ADB}.$$