МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

**Є. Ю. БЕРКУНСЬКИЙ, Т. Г. СМИКОДУБ,
А. Ю. ПАВЛЕНКО**

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни "Алгоритмізація та програмування"

У двох частинах

Частина 1

Рекомендовано Методичною радою НУК

Електронне видання
комбінованого використання на DVD-ROM

МИКОЛАЇВ • НУК • 2016

УДК 004.43(076)
ББК 32.973.2я73
Б 48

Автори:
Є. Ю. Беркунський, старш. викладач;
Т. Г. Смикодуб, старш. викладач;
А. Ю. Павленко, викладач

Рецензент К. В. Кошкін, д-р техн. наук, професор

Беркунський Є. Ю.

Б 48 Методичні вказівки до виконання лабораторних робіт з дисципліни "Алгоритмізація та програмування" : у 2 ч. Ч. 1 / Є. Ю. Беркунський, Т. Г. Смикодуб, А. Ю. Павленко. – Миколаїв : НУК, 2016. – 65 с.

Наведено теоретичні відомості та довідкову інформацію для виконання завдань лабораторних робіт з дисципліни: основні відомості про середовище розробки JetBrains CLion, структуру програм мовами C/C++, оператори, службові слова, інші елементи мови програмування. Визначено завдання та контрольні питання для лабораторних робіт.

Призначено для студентів спеціальності 6.050101 "Інформаційні управляючі системи та технології" всіх форм навчання. Можуть бути використані студентами інших спеціальностей.

УДК 004.43(076)
ББК 32.973.2я73

Навчальне видання

БЕРКУНСЬКИЙ Євген Юрійович
СМИКОДУБ Тетяна Георгіївна
ПАВЛЕНКО Альона Юріївна

**МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни
"Алгоритмізація та програмування"**

У двох частинах

Частина 1

Комп'ютерне верстання *А. Й. Лихіна*
Коректор *М. О. Паненко*

Формат 60×84/16. Ум. друк. арк. 3,8. Обсяг даних 10239 кб. Тираж 15 прим.
Вид. № 52. Зам. № 66.

Видавець і виготівник Національний університет кораблебудування імені адмірала Макарова
просп. Героїв Сталінграда, 9, м. Миколаїв, 54025, e-mail : publishing@nuos.edu.ua

Свідоцтво суб'єкта видавничої справи ДК № 2506 від 25.05.2006 р.

ВСТУП

Методичні рекомендації призначені для виконання лабораторних робіт з навчальної дисципліни "Алгоритмізація та програмування" у першому семестрі. Перед виконанням кожної роботи студенту необхідно вивчити відповідний матеріал лекції та звернути особливу увагу на теоретичні відомості, що передують опису лабораторних завдань. Наведені приклади програм не слід розглядати як єдиний можливий варіант розв'язання завдань.

Методичні рекомендації містять опис 7 лабораторних робіт. Кожен розділ відповідає окремій лабораторній роботі та складається з таких підрозділів: мета роботи й вимоги до теоретичної та практичної підготовки, що необхідна для виконання лабораторної роботи; рекомендації щодо підготовки до виконання лабораторної роботи; основні теоретичні відомості, необхідні для її виконання; суть роботи – загальна постановка завдання до лабораторної роботи (необов'язково); індивідуальні варіанти завдань; контрольні питання. При проведенні всіх лабораторних робіт використовується єдина конфігурація програмно-апаратних засобів: персональна ЕОМ типу IBMPC з процесором класу Intel Core i3 або вище, операційна система Windows 7 (або більш нова) або Linux (наприклад, Linux Mint), середовище програмування JetBrains CLion. Під час проведення лабораторних робіт студент повинен продемонструвати: творчий, індивідуальний підхід до розробки проектів (програмного коду); грамотне використання існуючого програмного забезпечення; навики програмування на мовах високого рівня C/C++. Студент повинен вміти

перетворити свою програму в програмний продукт, використовувати якісний аналіз програми, виконувати оцінку отриманих результатів. Велике значення має зручний інтерфейс з користувачем і поясненнями до програми.

Варіант завдання до лабораторної роботи обирається відповідно до номера студента в журналі групи або визначається викладачем.

Типовий порядок і методичні рекомендації до виконання роботи:

1. Уважно ознайомитися з методичними рекомендаціями до конкретної лабораторної роботи (теоретичними відомостями, прикладами, формулюванням завдань); використовуючи засоби середовища розробки, створити мінімальне консольне застосування.

2. Доповнити мінімальне застосування конкретним змістом відповідно до запропонованого завдання (див. приклади); відлагодити програму, усуваючи всі помилки, що виникли на етапі компіляції початкового тексту програми.

3. Виконання програми здійснити на даних, наведених у завданні до лабораторної роботи і на власних даних.

4. Знайти свою папку проекту й ознайомитися з її вмістом.

5. Виконати підсумковий запуск застосування за допомогою виконуваного модуля.

6. Відповісти на контрольні питання.

7. Оформити звіт і здати викладачеві.

Лабораторна робота № 1
Частина 1. Знайомство з інтегрованим середовищем розробки програм JetBrains CLion

Мета лабораторної роботи: набуття первинних практичних навичок роботи в середовищі програмування JetBrains CLion.

Перед виконанням лабораторної роботи студент повинен знати:

- призначення основних елементів вікна JetBrains CLion;
- основні елементи управління вікном JetBrains CLion;
- структуру та призначення основних елементів формату С-програми;
- призначення директив препроцесора, операторів програми і коментарів;
- призначення і способи використання динамічної довідки.

Після виконання лабораторної роботи студент повинен вміти:

- створювати проект із використанням стандартного шаблону;
- здійснювати введення й редагування тексту програми;
- знаходити та усувати помилки, які виникли на етапі написання коду програми.

Розробка простого консольного застосування

Запустіть JetBrains CLion, для чого виберіть відповідне посилання у Головному меню ОС, або двічі клацніть мишкою по ярлику застосування, якщо він виведений на робочий стіл. Після цього на екрані з'явиться вітальне (Welcome) вікно інтегрованого середовища розробки JetBrains CLion (рис. 1).

Для створення консольного застосування оберіть New Project. З'явиться діалогове вікно New Project (рис. 2).

У цьому діалоговому вікні вкажіть ім'я проекту (Project name:), оберіть місце розміщення проекту (Project location:) та натисніть OK. JetBrains CLion створить найпростіше застосування мовою C++, відкриє його у головному вікні (рис. 3).

Рисунок 1 – Вітальне вікно інтегрованого середовища розробки

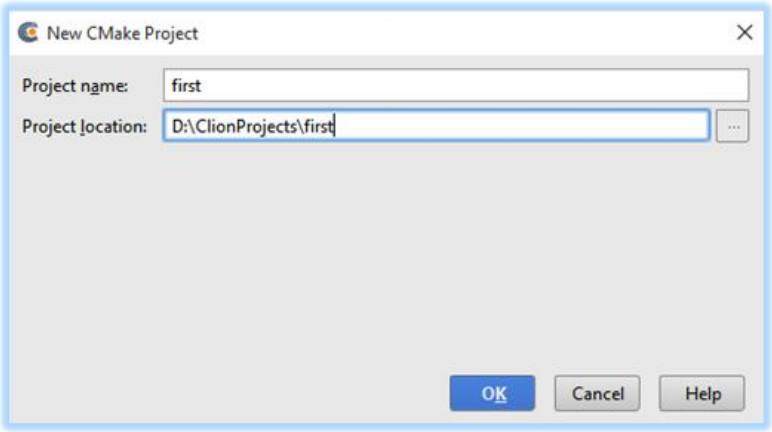


Рисунок 2 – Діалогове вікно New Project

Якщо шаблон застосування не було змінено, у текстовому редакторі коду програми буде відображено текст такої програми (якщо відображається інший – введіть цей):

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

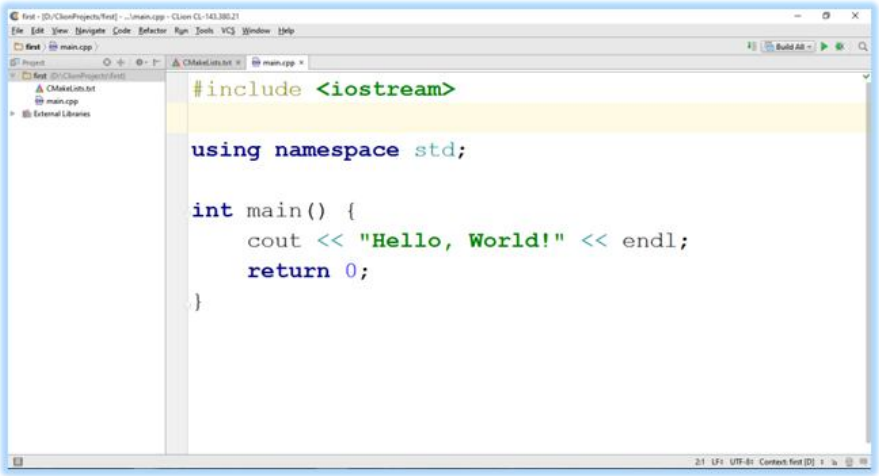


Рисунок 3 – Головне вікно інтегрованого середовища розробки

Відкомпілюйте програму і створіть виконуваний файл. Для створення виконуваного файлу рекомендується використовувати команду меню Run -> Build (або натиснути комбінацію клавіш Ctrl+F9). У каталозі, що був обраний як стандартний каталог виведення результатів збирання програми, буде створено виконуваний файл <ім'я_проекту>.exe. Для більш зручного налаштування цього каталогу можна скористатися діалогом File -> Settings… (рис. 4).

Натиснувши комбінацію клавіш Shift+F10 або кнопку "Run" (зелений трикутник) на панелі інструментів, запустіть створене застосування на виконання. У вікні виведення ви побачите результат роботи програми – вітання "Hello, World!" (рис. 5).

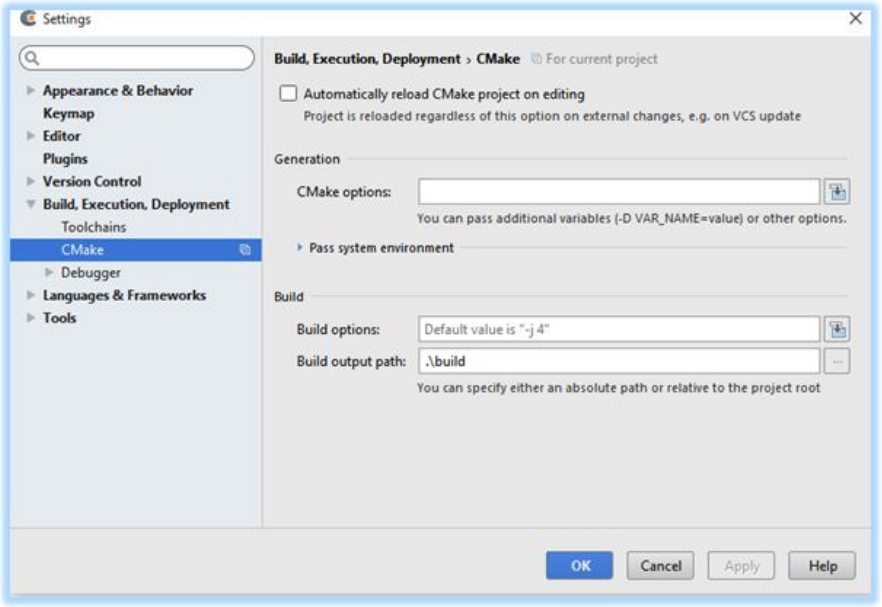


Рисунок 4 – Налаштування середовища розробки

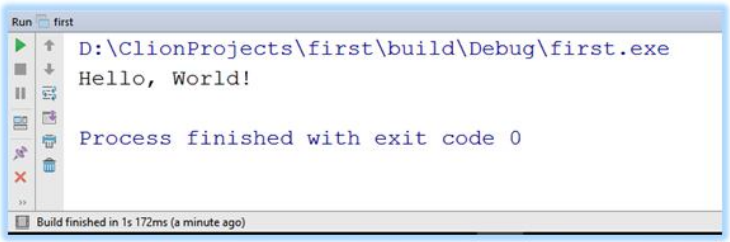


Рисунок 5 – Вікно виведення середовища JetBrains CLion

Порядок і методичні рекомендації до виконання роботи

1. Створити на робочому диску папку для проектів (ім'я папки повинне містити вичерпну інформацію про її користувача); запустити застосування JetBrains CLion.

2. Ознайомитися з елементами вікна JetBrains CLion.

3. Ознайомитися з командами кожного пункту головного меню.

4. Ознайомитися з елементами управління вікном JetBrains CLion.

5. Отримати у викладача навчально-демонстраційну програму або скористатися програмою, яка використовується в попередньому описанні середовища.

6. Виконати дії з введення, редагування, налагодження та збирання виконуваного модуля навчально-демонстраційної програми.

7. Підсумковий запуск виконуваного модуля виконати з командного рядка.

Контрольні запитання

1. Назвіть основні етапи розробки програми на ПЕОМ.
2. Назвіть основні елементи вікна застосування JetBrains CLion.
3. Назвіть основні органи управління вікном застосування JetBrains CLion.
4. Назвіть основні елементи структури програми мовами C/C++.
5. Перерахуйте основні операції з редагування тексту програми.
6. Чим відрізняється оператор мови програмування C/C++ від коментаря?
7. Яке завдання вирішується на етапі компіляції початкового тексту програми?
8. Яке завдання вирішується на етапі побудови виконуваного модуля програми?

Лабораторна робота № 1
Частина 2. Розв'язання задач лінійного характеру

Мета лабораторної роботи: отримання практичних навичок з підготовки, налагодження та виконання лінійних програм.

Перед виконанням лабораторної роботи студент повинен знати:

- класифікацію базових типів даних та їх основні характеристики;
- лексичні основи мови С++ – поняття: змінна, вираз, операнд, константа, оператор;
- пріоритети операцій;
- правила перетворення типів;
- основні бібліотечні математичні функції мови С++.

Після виконання лабораторної роботи студент повинен вміти:

- складати лінійні програми з використанням стандартних бібліотечних функцій;
- виконувати налагодження та покрокове тестування лінійних програм.

Короткі теоретичні відомості

1.1. Алфавіт мови С++, ідентифікатори та ключові слова

Для програмування завдань лінійного характеру (рис. 6), в яких операції виконуються в природному порядку, тобто в порядку їх запису в програмі, необхідно знати такі конструкції мови С++: ідентифікатори, службові слова, описи даних, вирази, оператори, вбудовані функції.

Вирази в мові С++ записуються за допомогою 26 рядкових та 26 прописних літер англійського алфавіту: abcdefghijkImnopqrstuvwxyz, ABCDEFGHIJKLMNOPQRSTUVWXYZ; десяти цифр: 0123456789; спеціальних символів: + – * / =, . _ : ; ? \ " '~ | ! # $ & ( ) [ ] { } ^ @. До спеціальних символів відноситься також пропуск. Комбінації деяких символів, не розділених пропусками, інтерпретуються як один значущий символ: ++ -- || && << >> >= <= == != += -= *= /= .?: :: /* */ //

Ідентифікаторами називаються імена, які надають змінним, константам, типам даних і функціям, які використовуються в програмах. Після опису ідентифікатора можна посилатися на об'єкт, що позначається ним.

Ідентифікатор – це послідовність символів довільної довжини, яка містить літери, цифри й символи підкреслення, обов'язково починається з літери або символу підкреслення.

У C++ враховується регістр букв. Компілятор сприймає прописні і рядкові букви як різні символи. Так, змінні userName та UserName розглядаються як два різні ідентифікатори.

Використання символу підкреслення на початку імені ідентифікатора не рекомендується. Два символи підкреслення (__) на початку імені ідентифікатора застосовуються в стандартних бібліотеках мови C++.

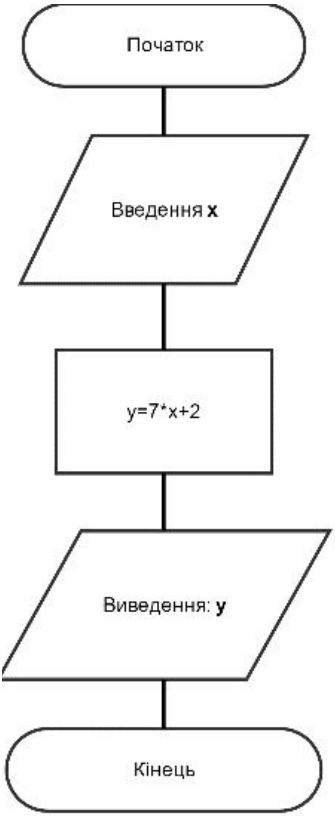


Рисунок 6 – Лінійний алгоритм

Ключові слова є зарезервованими ідентифікаторами, кожному з яких відповідає певна дія. Змінити призначення ключового слова не можна (директива препроцесора #define дозволяє створити "псевдонім" ключового слова, який дублює його дії, можливо, з деякими змінами). Імена ідентифікаторів, що створюються в програмі, не повинні збігатися з ключовими словами мов C/C++ (табл. 1).

1.2. Стандартні типи даних, модифікатори, кваліфікатори доступу й перетворення типів

Кожна програма обробляє певну інформацію. У C++ дані мають один з базових типів: **char** (текстові дані), **int** (цілі числа), **float** (числа з плаваючою точкою одинарної точності), **double** (числа з плаваючою точкою подвійної точності), **void** (порожні значення), **bool** (логічні значення), перерахування й покажчики.

Таблиця 1 – Ключові слова мови C/C++

alignas (починаючи з C++11) alignof (починаючи з C++11) and and_eq asm auto(1) bitand bitor bool break case catch char char16_t (починаючи з C++11) char32_t (починаючи з C++11) class compl const constexpr (починаючи з C++11) const_cast continue decltype (починаючи з C++11) default(1) delete(1) do double dynamic_cast else	enum explicit export extern false float for friend goto if inline int long mutable namespace new noexcept (починаючи з C++11) not not_eq nullptr (починаючи з C++11) operator or or_eq private protected public register reinterpret_cast	return short signed sizeof static static_assert (починаючи з C++11) static_cast struct switch template this thread_local (починаючи з C++11) throw true try typedef typeid typename union unsigned using(1) virtual void volatile wchar_t while xor xor_eq

Текстом (тип даних **char**) є один символ. Зазвичай кожен символ займає 8 біт або один байт з діапазоном значень від 0 до 255.

Цілі числа (тип даних **int**) знаходяться в діапазоні від –32 768 до 32 767 і займають 16 біт, тобто два байти, або одне слово. У Windows NT і Windows XP і сучасніших ОС Windows використовуються 32-розрядні цілі числа, що дозволяє розширити діапазон значень від –2 147 483 648 до 2 147 483 647.

У С++ підтримуються три типи цілих чисел. Разом зі стандартним типом **int** існують типи **short int** (коротке ціле) і **long int** (довге ціле).

Допускається скорочений запис **short** і **long**.

Числа з плаваючою точкою одинарної точності (тип даних **float**) можуть бути представлені як у фіксованому форматі, так і в експоненціальному. Діапазон значень – від ±3.4E-38 до ±3.4E+38, розмірність – 32 біта, тобто 4 байти або 2 слова.

Числа з плаваючою комою подвійної точності (тип даних **double**) мають діапазон значень від ±1.7E-308 до ±1.7E+308 і розмірності 64 біта, тобто 8 байтів або 4 слова. Раніше існував тип **long double** з розмірністю 80 біт і діапазоном від ±1.18E-4932 до ±1.18E+4932. У нових 32 розрядних версіях компіляторів він еквівалентний типу **double** і підтримується з міркувань зворотної сумісності з написаними раніше програмами.

Перерахування представляються кінцевим набором іменованих констант різних типів.

Тип даних **void**, як правило, застосовується у функціях, що не повертають ніякого значення. Цей тип даних також можна використовувати для створення узагальнених покажчиків.

Покажчики, на відміну від змінних інших типів, містять не дані в звичайному розумінні цього слова, а адреси пам'яті, де зберігаються дані.

Змінні нового логічного типу даних **bool** в С++ можуть містити тільки одну з двох констант: **true** або **false**.

Компілятор мови С++ дозволяє при описанні змінних вказувати модифікатор **unsigned**. Він застосовується з чотирма типами даних: **char**, **short**, **int** і **long**. Наявність даного модифікатора вказує на те, що значення змінної повинне інтерпретуватися як беззнакове число, тобто найстарший біт є бітом даних, а не бітом знака. Існує модифікатор **signed**, який виконує протилежну **unsigned** дію.

У С++ використовуються два кваліфікатори доступу **const** і **volatile**. Вони застосовуються для позначення незмінних змінних (**const**) і змінних, значення яких можуть змінитися в будь-який момент (**volatile**).

Іноді потрібно, щоб значення змінної залишалося постійним протягом всього часу роботи програми. Такі змінні називаються константними. Наприклад, якщо в програмі обчислюється довжина кола або площа круга, часто доводиться оперувати числом 3,14159.

Застосування кваліфікатора **volatile** може знадобитися в тому випадку, коли змінна оновлюється системними пристроями, наприклад,

таймером. При отриманні сигналу від таймера виконання програми переривається і значення змінної змінюється.

Кваліфікатор **volatile** також застосовується при описі об'єктів даних, спільно використовуваних різними процесами в багатозадачному середовищі.

У С++ застосовуються й інші кваліфікатори. Але їхнє використання виходить за рамки цієї дисципліни та методичних рекомендацій.

Часто буває, коли в операції беруть участь змінні різних типів. Такі операції називаються змішаними. Наприклад:

```
int a=2;
float res=3.7;
a=a*res; //a=2*3.7=7
```

У процесі виконання змішаних операцій компілятор автоматично проводить перетворення типів даних. Цілочисельне значення змінної a зчитується з пам'яті, приводиться до типу з плаваючою точкою та помножується на початкове значення змінної res, отримуємо 7,4. Результат у вигляді значення з плаваючою точкою присвоюється змінній цілого типу a, отримуємо 7. Автоматичні перетворення типів даних при виконанні змішаних операцій здійснюються відповідно до ієрархії перетворень. Суть полягає в тому, що з метою підвищення продуктивності в змішаних операціях значення різних типів тимчасово приводяться до того типу даних, який має більший пріоритет в ієрархії. Нижче перераховані типи даних у порядку зниження пріоритету: **double**, **float**, **long**, **int**, **short**.

Якщо значення перетвориться на тип, що має більшу розмірність, не буде мати місця втрата інформації, унаслідок чого не страждає точність обчислень.

Іноді потрібно змінити тип змінної, не чекаючи автоматичного перетворення. Для цього призначена операція приведення типу. Якщо в програмі необхідно тимчасово змінити тип змінної, потрібно перед її ім'ям ввести в круглих дужках назву відповідного типу даних. Наприклад:

```
r=v+(float)a/b;
r=v+a/(float)b;
r=v+(float)a/(float)b;
```

У всіх трьох випадках перед виконанням ділення відбувається явне приведення значення однієї або двох змінних до типу float.

1.3. Операції

С++ включає побітові операції, операції інкрементування й декрементування, умовну операцію, операцію кома, операції комбінованого присвоєння.

Побітові операції працюють зі змінними як з наборами бітів, а не як із числами. Ці операції використовуються в тих випадках, коли необхідно отримати доступ до окремих біт даних (наприклад, при виведенні графічних зображень на екран). Побітові операції застосовуються тільки до цілочисельних значень. На відміну від логічних операцій, з їх використанням порівнюються не два числа цілком, а окремі їх біти. Основні побітові операції: "І" (&), "АБО" (|) і "Виключне АБО" (^). Сюди можна також зарахувати унарну операцію побітового інвертування (~), яка інвертує значення бітів числа (табл. 2).

Таблиця 2 – Побітові операції "&", "|", "^"

a	B	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Операція & записує в біт результату одиницю тільки в тому випадку, якщо обидва порівнюваних біта дорівнюють 1. Ця операція часто використовується для маскування окремих бітів числа. Наприклад: 0xF1 & 0x35 = 0x31.

Операція | записує в біт результату одиницю в тому випадку, якщо хоча б один з порівнюваних бітів дорівнює 1. Ця операція часто застосовується для установки окремих бітів числа. Наприклад: 0xF1 | 0x35 = 0xF5.

Операція ^ записує в біт результату одиницю в тому випадку, якщо порівнювані біти відрізняються один від одного. Ця операція часто застосовується при виведенні зображень на екран, коли відбувається накладення декількох графічних шарів. Наприклад: 0xF1 ^ 0x35 = 0xC4.

У С++ існує дві операції зсуву: << – зсув ліворуч, >> – зсув праворуч. Дія першої операції полягає у зсуві бітового представлення цілочисельної змінної, вказаної зліва від операції, ліворуч на кількість бітів, задану праворуч від операції. При цьому звільнені молодші біти заповнюються нулями, а відповідна кількість старших бітів втрачається.

Зсув беззнакового числа на одну позицію ліворуч із заповненням молодшого розряду нулем еквівалентний множенню числа на 2. Наприклад:

```
unsigned int a = 65;    // молодший байт: 01000001
a <<= 1;                 // молодший байт: 10000010
cout << a;              // буде виведене 130
```

Зсув праворуч супроводжується аналогічними діями, тільки бітове представлення числа зрушується на вказану кількість бітів вправо. Значення молодших бітів втрачаються, а старші біти, що звільнилися, заповнюються нулями, якщо операнд беззнаковий, і значенням знакового біта інакше. Таким чином, зсув беззнакового числа на одну позицію праворуч еквівалентне діленню числа на два:

```
unsigned int a = 10;    // молодший байт: 00001010
a >>= 1;                // молодший байт: 00000101
cout << a;             // буде виведене 5
```

Збільшення (зменшення) значення змінної на 1 дуже часто зустрічається в програмах, тому розробники мови С++ передбачили для цих цілей спеціальні операції інкрементування (++) і декрементування (--). Так, замість рядка a+1 можна ввести рядок a++ або ++a.

За ситуації, коли операція ++ є єдиною у виразі, не має значення місце її розташування: до імені змінної або після нього. Значення змінної в будь-якому випадку збільшиться на одиницю.

У процесі роботи зі складними виразами необхідно уважно стежити, коли саме відбувається модифікація змінної. Потрібно розрізняти префіксні й постфіксні операції, які ставляться відповідно до або після імені змінної.

Наприклад, при постфіксному інкрементуванні i++ спочатку повертається значення змінної, після чого воно збільшується на одиницю. З іншого боку, операція префіксного інкрементування ++i вказує, що спочатку слід збільшити значення змінної, а потім повернути його як результат. Наприклад: нехай i=3, тоді

```
k=++i;  // набувають значення i=4, k=4
k=i++; //спочатку k=4, потім i збільшиться на одиницю (i=5)
```

```
k=--i;   //спочатку зменшиться на одиницю i=4, k=4
k=i--;   //k=4,i=3
```

У С++ представлені всі стандартні арифметичні операції: складання (+), віднімання (–), множення (*), ділення (*/) і ділення по модулю (%). Перші чотири операції не вимагають роз'яснень. Суть операції ділення по модулю:

```
int a=3, b=8, d;
d=b % a;   // результат: 2.
```

При діленні по модулю повертається залишок від операції цілочисельного ділення.

У С++ операція присвоювання (=) може входити до складу інших виразів. У результаті виконання операції присвоювання повертається значення, присвоєне лівому операнду. Наприклад, такий вираз цілком коректний:

```
v=8*(w=5); // v=40.
```

У даному випадку спочатку змінній w присвоюється значення 5, після чого це значення помножується на 8, а результат присвоюється змінній v. Комбіновані операції присвоєння наведені у табл. 3.

Таблиця 3 – Комбіновані операції присвоювання

Початковий оператор	Еквівалент	Коментар
`var=var+3;`	`var+=3;`	До змінної додається 3
`var=var-10;`	`var-=10;`	Із змінної віднімається 10
`var=var*3.14;`	`var*=3.14;`	Змінна помножується на 3.14
`var=var/2.5;`	`var/=2.5;`	Змінна ділиться на 2.5
`var=var&0xF;`	`var&=0xF;`	У змінній залишаються тільки 4 молодших розряди
`var=var|0xF;`	`var|=0xF;`	У змінній встановлюються 4 молодших розряди
`var=var<<3;`	`var<<=3;`	Змінна зсувається ліворуч на 3 розряди
`var=var>>5;`	`var>>=5;`	Змінна зсувається праворуч на 5 розрядів
`var=var%2;`	`var%=2;`	Взяття залишку при діленні var на 2
`var=var+1;`	`var++;`	Операція інкремента
`var=var-1;`	`var--;`	Операція декремента

Операції порівняння призначені для перевірки рівності або нерівності порівнюваних операндів. Усі вони повертають true у разі встановлення істинності виразу і false інакше.

Нижче перераховані оператори порівняння, використовувані в мовах С і С++ (табл. 4).

Таблиця 4 – Оператори порівняння

Операція	Виконувана перевірка
==	Дорівнює
!=	Не дорівнює
>	Більше
<	Менше
<=	Менше або дорівнює
>=	Більше або дорівнює

Логічні операції І (&&), АБО (||) і НЕ (!) повертають значення **true** або **false** залежно від логічного відношення між їх операндами. Так, операція && повертає **true**, коли істинні (не дорівнюють нулю) обидва його аргументи. Оператор || повертає **false** тільки в тому випадку, якщо обидва його аргументи не є істинними (дорівнюють нулю). Оператор ! інвертує значення свого операнду з **false** на **true** і навпаки.

Послідовність виконання різних операцій визначається компілятором.

Якщо не враховувати порядок розбору виразу компілятором, можуть бути отримані неправильні результати.

У табл. 5 перераховані всі операції мови С++ в порядку зниження їх пріоритету і вказаний напрям обчислення операндів (асоціативність): зліва направо або справа наліво.

Таблиця 5 – Пріоритет операцій (від високого до низького)

Операція	**Опис**	**Асоціативність**
1	2	3
++	Постфіксний (префіксний) інкремент	Зліва направо
--	Постфіксний (префіксний) декремент	
()	Виклик функції	
[]	Доступ до елемента масиву	
->	Непрямий доступ до члена класу	
.	Прямий доступ до члена класу	
!	Логічне НЕ	
~	Побітове НЕ	
-	Унарний мінус	
+	Унарний плюс	
&	Узяття адреси	

Продовж. табл. 5

1	2	3
*	Розкриття покажчика	
sizeof	Отримання розмірності виразу в байтах	
new	Динамічне створення об'єкта	
delete	Динамічне видалення об'єкта	
(тип даних)	Приведення типу	
.*	Прямий доступ до покажчика на член класу (через об'єкт)	Зліва направо
*	Множення	Зліва направо
/	Ділення	
%	Ділення по модулю	
+	Складання	Зліва направо
-	Віднімання	
<<	Зсув ліворуч	Зліва направо
>>	Зсув праворуч	
<	Менше	Зліва направо
>	Більше	
<=	Менше або дорівнює	
>=	Більше або дорівнює	
==	Дорівнює	Зліва направо
!=	Не дорівнює	
&	Побітове І	Зліва направо
^	Що побітове виключає АБО	Зліва направо
|	Побітове АБО	Зліва направо
&&	Логічне І	Зліва направо
II	Логічне АБО	Зліва направо
?:	Умовний вираз	Справа наліво
=	Просте присвоювання	Справа наліво
*=	Присвоювання з множенням	
/=	Присвоювання з діленням	
%=	Присвоювання з діленням по модулю	
+=	Присвоювання зі складанням	
-=	Присвоювання з відніманням	
<<=	Присвоювання із зсувом ліворуч	
>>=	Присвоювання із зсувом управо	
&=	Присвоювання з побітовим І	
|=	Присвоювання з побітовим АБО	
^=	Присвоювання з побітовим виключним АБО	
,	Кома	Зліва направо

У мовах C/C++ усі стандартні функції знаходяться у бібліотеках, які можна підключити за допомогою заголовочних файлів. Так, функції введення-виведення у стилі мови C описані у файлі stdio.h (cstdio). Функції та класи для введення-виведення у стилі C++ описані у файлах iostream (для введення-виведення із використанням стандартного пристрою) та fstream (для файлового введення-виведення). Обчислення у програмах на C/C++ неможливі без використання математичних функцій, які описані у файлі cmath (math.h).

Розглянемо приклад. Знаходження значення похідної функції в точці.

Постановка завдання: Задана функція $y = \sin(x)$. Знайти її похідну в точці $x = \pi/2$.

Для знаходження похідної в точці використовується відомий вираз:

$$y'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

```
#include <math.h>
#include <iostream>
using namespace std;
int main() {
    // Вибираємо приріст аргументу
     double dx=1.0e-11;
    // Вибираємо точку для обчислення похідної
    double x = 3.1415926;
    // Обчислюване значення функції в точці x+dx
    double f1=sin(x+dx);
    // Обчислюване значення функції в точці x
    double f2=sin(x);
    // Знаходимо значення похідної
     double pf=(f1-f2) /dx;
     cout << "dsin(x) /dx=" << pf <<" x= "<<x;
     return 0;
}
```

Завдання: Виконати практичну частину лабораторної роботи, яка наведена у додатку А.

Контрольні питання

1. Пояснить сенс поняття "оператор".
2. Що розуміється під типом даних?
3. Яка інформація повідомляється компілятору при оголошенні змінних і констант?
4. Дайте визначення виразу.
5. Вкажіть правила обчислення виразів.
6. Наведіть приклади операцій з однаковим пріоритетом.
7. Вкажіть операції з найвищим і найменшим пріоритетом.
8. Перерахуйте ключові слова, використовувані при оголошенні стандартних типів даних.

Лабораторна робота № 2
Підготовка і розв'язання на ПЕОМ задач з розгалуженням

Мета лабораторної роботи: набуття практичних навичок з підготовки, налагодження і виконання програм, що розгалужуються.

Перед виконанням лабораторної роботи студент повинен знати:

- методику розробки програми із загальною лінійною частиною й декількома гілками;
- алгоритми виконання й синтаксис операторів if, if/else, switch/case і умовної операції ?:.

Після виконання лабораторної роботи студент повинен вміти:

- розробляти і налагоджувати програми з розгалуженнями.

Короткі теоретичні відомості

У багатьох випадках подальше виконання програми від деякої точки повинне йти різними шляхами залежно від певних умов.

Наприклад, при настисканні мишею однієї кнопки програма повинна приступити до виконання гілки 1, а при натисканні на іншу кнопку – до виконання гілки 2. Такі умовні переходи зовсім не обов'язково пов'язані з командами користувача; програма може сама вибрати подальший шлях, наприклад, за наслідками деякої математичної операції: при нульовому результаті виконати один фрагмент, при негативному – другий, при позитивному – третій. У C++ існує три базових оператори вибору: if, if/else, switch/case і умовної операції ? : .

Оператор if

Оператор if призначений для виконання команди або блоку команд залежно від того, істинно задана умова, чи ні. Формат оператора if:

```
if (умова) вираз;
```

Якщо в результаті перевірки умови повертається значення true, виконується вираз, після чого управління передається наступному рядку

програми. Якщо ж результатом перевірки умови є значення false, вираз пропускається.

Оператор if/else дозволяє вибірково виконувати одну з двох дій залежно від умови. Формат даної інструкції має вигляд:

```
if (умова) вираз 1; else вираз 2;
```

Якщо в результаті перевірки умови повертається значення true, виконується вираз 1, інакше – вираз 2.

Якщо операторна частина гілки **if** або **else** містить не один вираз, а декілька, необхідно укласти їх у фігурні дужки. Після закриваючої фігурної дужки крапка з комою не ставиться.

Оператор **if** обох форм реалізують алгоритми, представлені на рис. 7.

Як аналізований вираз в операторові if найчастіше використовується одна з операцій відношення.

Рисунок 7 – Алгоритми роботи операторів if

Разом з операціями відношення в інструкції if широко використовуються логічні операції. Об'єднуючи їх з операціями відношення, можна створювати комбіновані конструкції перевірки даних.

Оператор switch/case

Оператор switch/case дозволяє залежно від значення деякого виразу вибрати один з багатьох варіантів продовження програми.

Оператор має такий формат:

```
switch(вираз) {
   case  значення 1: оператор 1; break;
   case  значення 2: оператор 2; break;
...
   case  значення N: оператор N; break;
   default: оператор
N+1;
}
```

Оператор switch/case реалізує алгоритм, наведений на рис. 8.

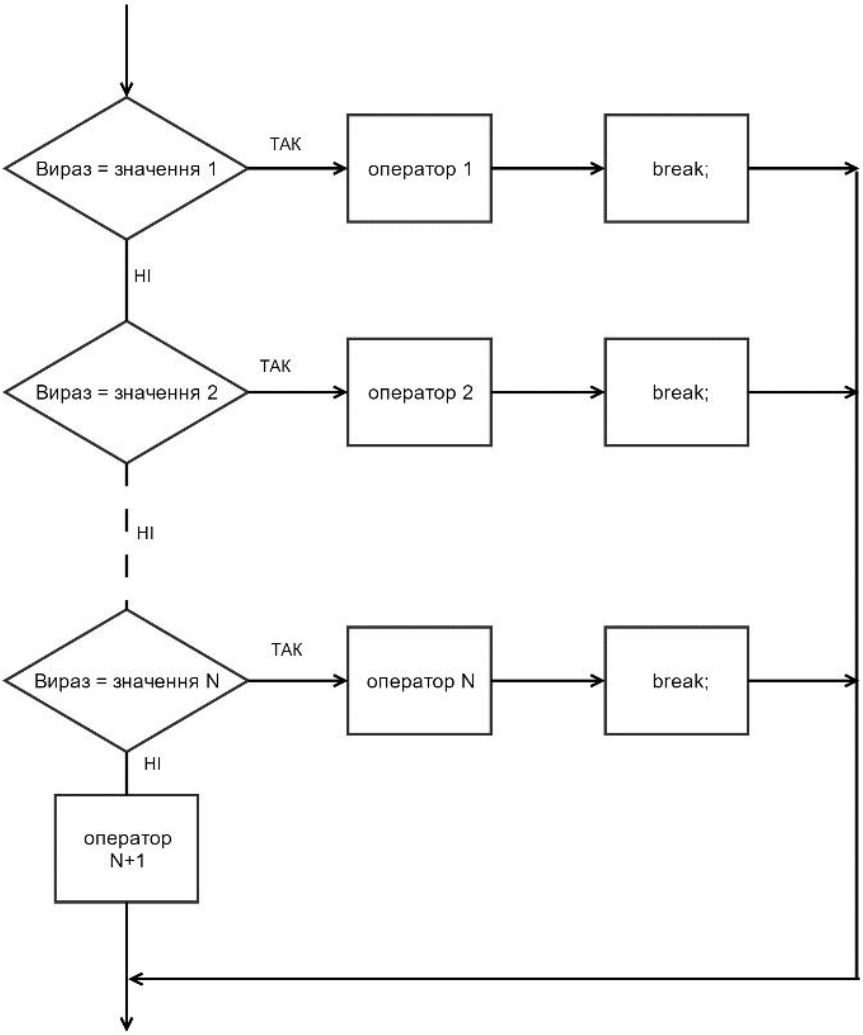


Рисунок 8 – Алгоритми роботи оператора switch/case

Оператор switch/case може бути використаний у варіанті без оператора N+1.

Як вираз при операторові switch зазвичай використовується змінна типу int або char, хоча можна використовувати й складніші вирази, в які

входять, наприклад, арифметичні або логічні операції над декількома змінними та константами.

Як значення при операторові case зазвичай використовуються просто константи (у числовій формі або в символьній, якщо вони були заздалегідь визначені за допомогою оператора препроцесора #define), проте можуть використовуватися і вирази над константами.

Виконання оператора switch починається з обчислення виразу в дужках, який повинен давати цілочисельний результат. Цей результат послідовно порівнюється із значеннями при операторах case, і, якщо буде виявлено рівність результатів, то виконується оператор відповідного case. Якщо збіги результатів не виявлено, виконується оператор при операторові default, якщо оператор default відсутній, то починають виконуватися оператори, наступні за всією конструкцією switch/case.

Оператори break, continue і goto

Оператор break використовується для виходу з оператора while, do.while, for і switch, що безпосередньо його містить. Управління передається на оператора, наступного за оператором, з якого здійснюється вихід. Приклад додатка оператора break наведений вище.

Оператор continue використовується для пропускання частини виконуваної ітерації циклу, який безпосередньо його містить, що залишилася. Якщо умовами циклу допускається нова ітерація, то вона виконується, інакше цикл завершується.

Оператор goto реалізує безумовний перехід, тобто дозволяє перейти в будь-яку точку програми, як вперед по тексту програми, так і назад. Точка переходу позначається за допомогою мітки, яка є довільним ідентифікатором з двокрапкою в кінці.

***Завдання*:** Виконати практичну частину лабораторної роботи, яка наведена у додатку А.

Контрольні питання

1. Що таке обчислювальний процес, який розгалужується?
2. Які форми запису має умовний оператор if?
3. Назвіть відмітні особливості умовного виразу порівняно з умовним оператором.
4. Напишіть програму пошуку мінімального числа з трьох заданих чисел.

Лабораторна робота № 3
Підготовка і розв'язання на ПЕОМ задач з використанням циклів

Мета лабораторної роботи: набуття практичних навиків з підготовки, відлагодження та виконання циклічних програм.

Перед виконанням лабораторної роботи студент повинен знати:

- основи застосування стандартних операторів циклу: while, do while, for.

Після виконання лабораторної роботи студент повинен вміти:

- розробляти типові циклічні програми на мові С++.

Короткі теоретичні відомості

Оператори циклу використовують для виконання деякого фрагмента програми кілька разів. В окремих випадках фрагмент виконується в кожному послідовному кроці циклу без змін; частіше кожен крок циклу дещо відрізняється від попереднього.

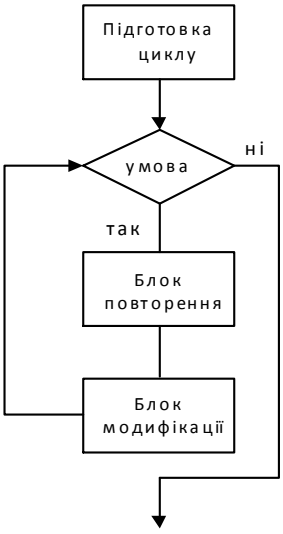


Рисунок 9 – Схема алгоритму циклічної структури

Для грамотної реалізації будь-якого циклічного обчислювального процесу необхідно виконати дії, представлені у вигляді узагальненого алгоритму на рис. 9.

Підготовка циклу полягає у визначенні початкових значень змінних, що будуть змінюватися у циклі, до початку виконання циклу.

Блок повторення – це дії, які повторюються в циклі. Вони завжди однакові, при цьому їх багаторазове повторення здійснюється при різних значеннях змінних циклу.

Модифікація – це зміна значень змінних циклу перед кожним новим повторення циклу.

Блок повторення та Блок модифікації разом складають тіло циклу.

Умова продовження циклу (команда переходу) полягає в перевірці умови продовження

або закінчення циклу, тобто визначає, скільки разів потрібно повторити тіло циклу.

Існує три види циклів: while, for і do…while.

Оператор циклу while називається циклом з передумовою та має такий формат:

```
while (вираз) тіло циклу;
```

Оператор while реалізує алгоритм, представлений на рис. 10.

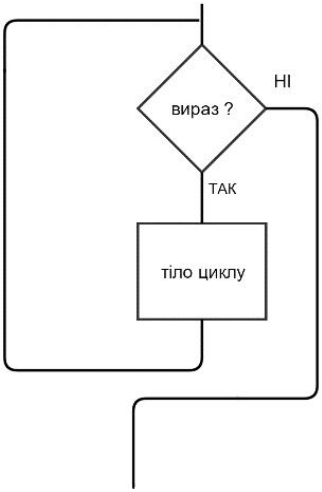


Рисунок 10 – Цикл while

Як вираз допускається використовувати будь-який вираз мови С++, а як тіло – будь-який оператор, зокрема порожній або складений. Схема виконання оператора while така:

1. Обчислюється вираз.

2. Якщо вираз false, то виконання оператора while закінчується і виконується наступний за порядком оператор. Якщо вираз true, то виконується тіло циклу.

3. Процес повторюється з пункту 1.

4. Тіло циклу виконується доти, поки значення виразу рівне true.

5. Вираз обчислюється перед кожним виконанням оператора.

Цикл for має такий формат:

```
for (вираз 1; вираз 2; вираз 3;) тіло циклу;
```

Оператора for реалізує алгоритм, представлений на рис. 11.

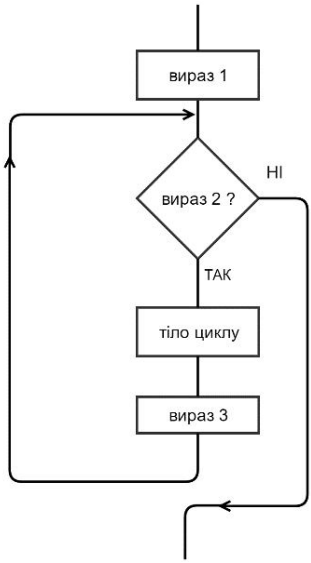


Рисунок 11 – Цикл for

Вираз 1 зазвичай використовується для встановлення початкового значення змінних, керівників циклом. Вираз 2 – це вираз, що визначає умову, при якій тіло циклу виконуватиметься. Вираз 3 визначає зміну змінних, керівників циклом після кожного виконання тіла циклу.

Схема виконання оператора for:

1. Обчислюється вираз 1.

2. Обчислюється вираз 2.

3. Якщо значення виразу 2 відмінне від нуля (true), виконується тіло циклу, обчислюється вираз 3 і здійснюється перехід до пункту 2. Якщо вираз 2 дорівнює нулю (false), то управління передається до оператора, наступного за оператором for.

Істотне те, що перевірка умови завжди виконується на початку циклу. Це означає, що тіло циклу може жодного разу не виконатися, якщо умова виконання відразу буде хибною.

Цикл for є зручним скороченим записом для циклу while вигляду

```
вираз 1;
while (вираз 2) {
   тіло циклу;
   вираз 3;
}
```

Вираз 1 задає початкові умови виконання циклу, вираз 2 забезпечує перевірку умови виходу з циклу, а вираз 3 модифікує умови, задані виразом 1. Будь-який з виразів може бути опущений. Якщо опущено вираз 2, то за замовчуванням замість нього підставляється значення true.

Наприклад, цикл: `for (;вираз 2;) тіло циклу;` з опущеними вираз 1 і вираз 3 еквівалентний циклу `while (вираз 2) тіло циклу;`

Цикл: `for (;;) тіло циклу;` зі всіма опущеними виразами еквівалентний циклу `while (true) тіло циклу;` тобто еквівалентний нескінченному циклу.

Такий цикл може бути перерваний тільки явним виходом з нього за допомогою операторів `break, goto` або `return`, що містяться в тілі циклу.

Оператор циклу do while називається оператором циклу з постумовою і використовується в тих випадках, коли необхідно виконати тіло циклу хоч би один раз. Формат оператора має такий формат:

```
do тіло циклу while (вираз);
```

Схема виконання оператора do while:

1. Виконується тіло циклу (яке може бути складеним оператором).

2. Обчислюється вираз.

3. Якщо вираз false, то виконання оператора do while закінчується й виконується наступний за порядком оператор. Якщо вираз true, то виконання оператора продовжується з пункту 1.

Щоб перервати виконання циклу до того, як умова стане хибною, можна використовувати оператора break.

Оператор do while реалізує алгоритм, наведений на рис. 12.

На відміну від циклу while, в якому перевірка умови закінчення циклу робиться до виконання тіла циклу, в циклі do while така перевірка має місце після виконання тіла циклу. Отже, тіло циклу do while буде виконано хоч би один раз, навіть якщо вираз має значення false із самого початку.

***Завдання*:** Виконати практичну частину лабораторної роботи, яка наведена у додатку А.

Рисунок 12 – Цикл do…while

Контрольні питання

1. Що таке циклічний обчислювальний процес?
2. Які оператори використовуються для організації циклів?
3. Опишіть ситуації, коли слід використовувати кожен з трьох операторів циклу.
4. Які три операції повинні проводитися в програмі при організації будь-якого циклу?
5. Охарактеризуйте переваги, які дає цикл for.

Лабораторна робота № 4
Підготовка і розв'язання на ПЕОМ задач з використанням функцій

Мета лабораторної роботи: вивчення особливостей використання функцій при вирішенні типових економічних завдань.

Після виконання лабораторної роботи студент повинен знати:

- структуру оголошення (прототипи) і визначення функцій;
- механізми передачі аргументів у функцію: за значенням, за посиланням, за покажчиком;
- механізми повернення функцією декількох значень.

Після виконання лабораторної роботи студент повинен вміти:

- застосовувати правила розробки і використання функцій різних типів при вирішенні типових економічних завдань;
- створювати власні заголовні файли з визначенням функцій користувача.

Короткі теоретичні відомості

Функція – це іменована послідовність описів і операторів, що виконує яку-небудь закінчену дію. Функція може приймати параметри і повертати значення.

Будь-яка програма на C++ складається з функцій, одна з яких повинна мати ім'я main (з неї починається виконання програми). Функція починає виконуватися у момент виклику. Будь-яка функція має бути оголошена і визначена. Як і для інших величин, оголошень може бути декілька, а визначення тільки одне.

Оголошення функції повинне знаходитися в тексті раніше її виклику для того, щоб компілятор міг здійснити перевірку правильності виклику.

Оголошення функції (прототип, заголовок, сигнатура) задає її ім'я, тип значення, що функція повертає і список параметрів.

Визначення функції містить, окрім оголошення, тіло функції, що є послідовністю операторів і описів у фігурних дужках:

```
 клас ]  тип ім'я  ( [ список_параметрів ] ) {
      // тіло функції
}
```

Тип значення, яке повертає функція, може бути будь-яким, окрім масиву і функції (але може бути покажчиком на масив або функцію). Якщо функція не повинна повертати значення, вказується тип void.

Список параметрів визначає величини, які потрібно передати у функцію при її виклику. Елементи списку параметрів розділяються комами. Для кожного параметра, що передається у функцію, вказується його тип та ім'я (в оголошенні імена можна опускати).

У визначенні, в оголошенні і при виклику однієї і тієї ж функції типи і порядок проходження параметрів повинні збігатися.

На імена параметрів обмежень по відповідності не накладається, оскільки функцію можна викликати з різними аргументами, а в прототипах імена ігноруються компілятором (вони використовуються тільки для поліпшення читаності програми).

Функцію можна визначити як вбудовану за допомогою модифікатора inline, який рекомендує компілятору замість звернення до функції поміщати її код безпосередньо в кожну точку виклику. Модифікатор inline ставиться перед типом функції. Він застосовується для коротких функцій, щоб понизити накладні витрати на виклик (збереження і відновлення регістрів, передача управління). Директива inline носить рекомендаційний характер і виконується компілятором у міру можливості. Використання inline-функцій може збільшити об'єм виконуваної програми. Визначення функції повинне передувати її викликам, інакше замість inline-розширення компілятор згенерує звичайний виклик.

Тип значення, що повертає функція, і типи параметрів спільно визначають тип функції.

Для виклику функції в простому випадку потрібно вказати її ім'я, за яким у круглих дужках через кому перераховуються імена аргументів, що передаються. Виклик функції може знаходиться в будь-якому місці програми, де по синтаксису допустимий вираз того типу, який формує функція. Якщо тип значення, яке повертає функція, не void, то вона може входити до складу виразу або, в окремому випадку, розташовуватися в правій частині оператора присвоювання.

Розглянемо приклад програми, що знаходить найменше з трьох чисел, використовуючи при цьому функцію min3(), викликаючи її за іменем.

```
#include <iostream>

int min3(int x, int y, int z);
```

```
using namespace std;
int main() {
    int a,b,c;
    cin >> a >> b >> c;
    int x = min3(a,b,c);
    cout << "min = " << x << endl;
    return 0;
}

int min3(int x, int y, int z) {
    return x < y ? x < z ? x : z : y < z ? y : z;
}
```

При виконанні програми можна отримати такий результат:

```
4 7 3
min = 3

Process finished with exit code 0
```

Особливості виконання функцій

Усі величини, які описані всередині функції, а також її параметри, є локальними. Зоною їх дії є функція. При виклику функції, як і при вході в будь-який блок, в стеку виділяється пам'ять під локальні автоматичні змінні. Крім того, в стеку зберігається вміст регістрів процесора на момент, що передує виклику функції, і адреса повернення з функції для того, щоб після виходу з неї можна було продовжити виконання функції, яка її викликала.

При виході з функції відповідна ділянка стека звільняється, тому значення локальних змінних між викликами однієї і тієї ж функції не зберігаються. Якщо цього потрібно уникнути, при оголошенні локальних змінних використовується модифікатор static (проте це робити не рекомендується).

Значення, що повертається функцією

Механізм повернення з функції у функцію, що викликала її, реалізується оператором

```
return [ вираз ];
```

Якщо функція описана як **void**, вираз не вказується. Оператор **return**; використовується для дострокового виходу з функції.

Оператор **return** можна опускати для функції типу **void**, якщо повернення з неї відбувається перед закриваючою фігурною дужкою, і для функції **main**.

Вираз, вказаний після **return**, неявно перетвориться до типу повертаного функцією значення і передається в точку виклику функції.

Функція може містити декілька операторів **return** (це визначається потребами алгоритму).

Обмін інформацією між функціями

При спільній роботі функції повинні обмінюватися інформацією. Це можна здійснити:

- за допомогою глобальних змінних;
- через параметри функції;
- через значення, що повертає функція.

Обмін інформацією за допомогою глобальних змінних

Глобальні змінні видно у всіх функціях, де не описані локальні змінні з тими ж іменами, тому використовувати їх для передачі даних між функціями дуже легко. Проте, це не рекомендується, оскільки затрудняє відлагодження програми і перешкоджає розміщенню функцій у бібліопапки загального користування. Потрібно прагнути до того, щоб функції були максимально незалежні, а їх інтерфейс повністю визначався прототипом функції.

Використання параметрів функції для обміну інформацією між функціями

Механізм параметрів є основним способом обміну інформацією між функціями, що викликаються і викликають. Параметри, перераховані в заголовку опису функції, називаються формальними, а записані в операторі виклику функції – фактичними.

При виклику функції насамперед обчислюються вирази, що стоять на місці фактичних параметрів; потім у стеку виділяється пам'ять під формальні параметри функції відповідно до їх типу, і кожному з них привласнюється значення відповідного фактичного параметра. При цьому перевіряється відповідність типів і при необхідності виконуються їх перетворення. При невідповідності типів видається діагностичне повідомлення.

Існує два способи передачі параметрів у функцію: за значенням і за адресою.

При передачі за значенням в стек заносяться копії значень фактичних параметрів, і оператори функції працюють з цими копіями. Доступу

до початкових значень параметрів у функції немає, а, отже, немає і можливості їх змінити.

При передачі за адресою в стек заносяться копії адрес параметрів, а функція здійснює доступ до елементів пам'яті за цими адресами і може змінити початкові значення параметрів.

Розглянемо приклад програми, що ілюструє перераховані способи передачі параметрів у функцію:

```
// Способи передачі параметрів у функцію
#include <iostream>
using namespace std;

void f(int i, int* j, int& k);
int main()
{
    int i = 1, j = 2, k = 3;
    cout << "i j k\n";
    cout << i << " " << j << " " <<  k << endl;
    f(i, &j, k);
    cout << i << " " <<  j << " " <<  k <<endl;
    return 0;
}

void f (int i, int* j, int& k)
{
    i++; (*j)++; k++;
}
```

Результат роботи програми

```
i j k
1 2 3
1 3 4

Process finished with exit code 0
```

Перший параметр (i) передається за значенням. Його зміна у функції не впливає на початкове значення. Другий параметр (j) передається за адресою за допомогою покажчика, при цьому для передачі у функцію адреси фактичного параметра використовується

операція взяття адреси, а для набуття його значення у функції потрібна операція розіменування. Третій параметр (k) передається за адресою за допомогою посилання.

При передачі по посиланню у функцію передається адреса вказаного при виклику параметра, а усередині функції всі звернення до параметра неявно "розіменуються". Тому використання посилань замість покажчиків покращує читабельність програми, позбавляючи від необхідності застосовувати операції отримання адреси і розіменування. Використання посилань замість передачі за значенням ефективніше, оскільки не вимагає копіювання параметрів, що має значення при передачі структур даних великого об'єму.

Якщо потрібно заборонити зміну параметра всередині функції, використовується модифікатор const, наприклад:

```
int f(const char*);
```

Рекомендується вказувати const перед всіма параметрами, зміна яких у функції не передбачена. Це полегшує відлагодження великих програм, оскільки по заголовку функції можна зробити висновок про те, які величини в ній змінюються, а які – ні. Крім того, на місце параметра типу const& може передаватися константа, а для змінної при необхідності виконуються перетворення типу.

Таким чином, початкові дані, які не повинні змінюватися у функції, переважно передавати їй за допомогою константних посилань. За замовчуванням параметри будь-якого типу, окрім масиву і функції (наприклад, дійсного, структурного, перерахування, об'єднання, покажчик), передаються у функцію за значенням.

Перевантаження функцій

У мові C++ передбачена можливість створення декількох функцій з однаковим ім'ям. Це називається перевантаженням функцій. Перевантажені функції повинні відрізнятися один від одного списками параметрів: або типом одного або декількох параметрів, або різною кількістю параметрів, або і тим і іншим одночасно. Розглянемо такий приклад:

```
int  func(int, int);
int  func(long, long);
int  func(long);
```

Функція **`func()`** перевантажена з трьома різними списками параметрів. Перша і друга версії відрізняються типами параметрів, а третя – їх кількістю.

Типи значень, що повертають перевантажені функції, можуть бути однаковими або різними. Слід мати на увазі, що при створенні двох функцій з однаковим ім'ям і однаковим списком параметрів, але з різними типами значень, що повертаються, призведе до помилки компіляції.

Перевантаження функцій також називається поліморфізмом функцій. Полі (гр. poly) означає багато, морфе (гр. morphe) – форма, тобто поліморфічна функція – це функція, що відрізняється різноманіттям форм.

Під поліморфізмом функції розуміють існування в програмі декількох перевантажених версій функції, що мають різні призначення. Змінюючи кількість або тип параметрів, можна привласнити двом або декільком функціям одне і те ж ім'я. При цьому ніякої плутанини при виклику функцій не буде, оскільки потрібна функція визначається по збігу використовуваних параметрів. Це дозволяє створити функцію, яка зможе, наприклад, усереднювати цілочисельні значення, значення типу double або значення інших типів без необхідності створювати окремі імена для кожної функції – Averageints(), AverageDoubles() і так далі.

Припустимо, потрібно створити функцію, яка подвоює будь-яке отримане нею значення. При цьому хотілося б мати можливість передавати їй значення типу int, long, float або double. Без перевантаження функцій довелося б створювати чотири різні функції:

```
int Doubleint(int);
long DoubleLong(long);
float DoubleFloat(float);
double DoubleDouble(double);
```

За допомогою перевантаження функцій можна використовувати такі оголошення:

```
int Double(int);
long Double(long);
float DouDle(float);
double Double(double);
```

Завдяки використанню перевантажених функцій не потрібно турбуватися про виклик у програмі потрібної функції, що відповідає типу змінних, які передаються. При виклику перевантаженої функції компілятор автоматично визначить, який саме варіант функції слід використовувати.

Параметри функції, використовувані за замовчуванням

Для кожного параметра, що оголошується в прототипі і визначенні функції, має бути передане відповідне значення у виклику функції.

Передаване значення повинне мати оголошений тип. Отже, якщо деяка функція оголошена як long func(int); то вона дійсно повинна набувати цілочисельного значення. Якщо тип оголошеного параметра не збіжиться з типом передаваного аргументу, компілятор повідомить про помилку.

З цього правила існує одне виключення, яке набуває чинності, якщо в прототипі функції для параметра оголошується стандартне значення. Це значення, яке використовується у тому випадку, якщо при виклику функції для цього параметра не встановлено ніякого значення. Дещо змінимо попереднє оголошення:

```
long func(int  x = 50);
```

Прототип такого оголошення потрібно розуміти таким чином. Функція func(int) повертає значення типу long і приймає параметр типу int. Але якщо при виклику цієї функції аргумент наданий не буде, використовуйте замість нього число 50. А оскільки в прототипах функцій імена параметрів не обов'язкові, то останній варіант оголошення можна переписати по-іншому:

```
long func (int = 50);
```

Визначення функції не змінюється при оголошенні значення параметра, що задається за замовчуванням. Тому заголовок визначення цієї функції виглядатиме так, як і раніше:

```
long func (int x);
```

Якщо при виклику цієї функції аргумент не встановлюється, то компілятор привласнить змінною x значення 50. Ім'я параметра, для якого в прототипі встановлюється значення за замовчуванням, може не збігатися з ім'ям параметра, що вказується у заголовку функції: значення, задане за замовчуванням, привласнюється за позицією, а не по імені.

Установку значень за замовчуванням можна призначити будь-яким або всім параметрам функції. Але одне обмеження все ж таки діє: якщо якийсь параметр не має стандартного значення, то жоден з попередніх по відношенню до нього параметрів також не може мати стандартного значення.

Припустимо, прототип функції має вигляд `long func (int Paraml, int Param2, int Param3);` тоді параметру Param2 можна призначити стандартне значення тільки в тому випадку, якщо призначене стандартне значення і параметру Pagam3. Параметру Paraml можна призначити стандартне значення тільки в тому випадку, якщо призначені стандартні значення як параметру Pagam2, так і параметру Pagam3.

***Завдання*:** Виконати практичну частину лабораторної роботи, яка наведена у додатку А.

Контрольні питання

1. Назвіть три умови, необхідні для використання функції.
2. Що таке прототип функції, для чого він потрібний?
3. Чим відрізняється оголошення функції від визначення функції?
4. Коли доцільно використовувати статичну змінну всередині функції?
5. Назвіть три способи передачі параметрів у функцію. Опишіть ситуації, коли слід використовувати конкретний чин. Наведіть приклади.
6. Що таке перевантаження функції, коли її доцільно використовувати?
7. Назвіть правила завдання параметрів функції за замовчуванням.
8. Як і в яких випадках створюються заголовні файли користувача?

Лабораторна робота № 5
Підготовка і розв'язання на ПЕОМ задач обробки одновимірних масивів

Мета лабораторної роботи: освоєння методики обробки інформації в одновимірних масивах.

Перед виконанням лабораторної роботи студент повинен знати:

- визначення і правила опису одновимірних масивів різних типів; способи доступу до елементів масиву даних;
- способи ініціалізації елементів масиву;
- способи роботи з одновимірними масивами;
- способи алгоритмізації пошуку мінімальних і максимальних значень в одновимірних масивах.

Після виконання лабораторної роботи студент повинен вміти:

- виконувати ініціалізацію масивів різних типів і розмірності;
- використовувати засоби мови С++ для перетворення типів даних;
- розробляти і виконувати програми пошуку мінімальних і максимальних значень в одновимірних масивах.

Короткі теоретичні відомості

Масив – це сукупність логічно взаємозв'язаних даних одного типу, об'єднаних загальним ім'ям. Масив у С++ – це набір однотипних даних (об'єктів), що мають загальне ім'я і що розрізняються місцеположенням у цьому наборі (або індексом, привласненому кожному елемента масиву).

У програмі можуть бути масиви цілих чисел або чисел з плаваючою точкою, символьні масиви і т. д.

Опис одновимірного масиву

Опис складається зі специфікації типу, ідентифікатора і розмірності масиву. Звернення до елементів масиву виконується за їх індексами, які завжди починаються від нуля. Якщо, наприклад, оголосити в програмі

масив на ім'я mas зі 100 цілих чисел `int mas[100];` то найперший елемент масиву отримує позначення mas[0], наступний – mas[1] і так далі до останнього елемента mas [99].

Так само можна оголосити масиви даних будь-яких інших типів:

```
char c[20];  //Масив з 20 символів
float f[16];  //Масив з 16 чисел з плаваючою точкою.
Після оголошення масиву його можна заповнити числами,
використовуючи індексний оператор [ ], наприклад:
int myArr[3];   // Оголошення масиву
myArr[0] = -12; // Ініціалізація першого елемента
                // (його індекс дорівнює нулю) значенням -12
myArr[1] = 5;
myArr[2] = 37;
```

Доступ до окремих елементів масиву здійснюється також за допомогою індексного оператора:

```
int result = myArr[0]  + myArr[1] + myArr[2];
```

Масив можна оголосити і заповнити одночасно, наприклад:

```
int myArr[3]  =  { -12, 5, 37 } ;
```

Якщо розмірність задана, то число значень в списку ініціалізації не повинне її перевищувати.

Якщо розмірність масиву більше числа значень в списку, то елементи масиву, що не ініціалізували явно, будуть встановлені в 0, наприклад:

```
int myArr[5]={ 0, 1, 2 }; //myArr буде рівне { 0, 1, 2,
0, 0 }
```

Значення розмірності має бути константним виразом, тобто розмірність має бути відома на етапі трансляції. Це означає, що змінна не може використовуватися для завдання розмірності масиву.

Якщо в масиві не дуже багато елементів та їх значення відомі заздалегідь, масив можна ініціалізувати разом з його оголошенням, уклавши перелік значень у фігурні дужки:

```
int ns[10]={0,1,2,3,4,5,6,7,8,9};
```

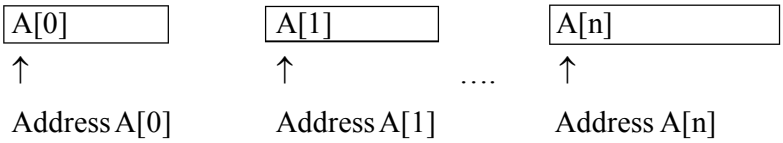
Якщо ж масив великий, заповнити його доведеться програмно в циклі. Нехай ми хочемо утворити масив зі 100 елементів, заповнених натуральним рядом чисел. Це робиться таким чином:

```
int mas[100];
for (int i=0;i<100;i++) mas[i]=i+1;
```

До елементів масиву можна звертатися і вибірково, наприклад **`test[50]=0x7FFA;`**

Збереження одновимірних масивів

У C++ одновимірний масив логічно зберігається як послідовність упорядкованих елементів у пам'яті. Кожен елемент відноситься до одного і того ж типу даних.



У C++ ім'я масиву є константою і розглядається як адреса першого елемента цього масиву. Так, в оголошенні `type A[arraySize];` ім'я масиву A є константою і визначає місцеположення в пам'яті першого елементу A[0]. Елементи A[1], A[2] і т.д. слідують за ним послідовно.

Припустимо, що sizeof(type) = M, тоді весь масив A займає M * arraySize байт.

Компілятор задає таблицю, названу дескриптором масиву, для ведення запису характеристик масиву.

Таблиця включає інформацію про розмір кожного елемента масиву, початкову адресу масиву і кількість елементів у цьому масиві:

Початкова адреса:	A
Кількість елементів масиву:	arraySize
Розмір типу:	M = sizeof(type)

Ця таблиця використовується компілятором також для реалізації функції доступу (access function), яка визначає адресу елемента в пам'яті.

Символьні масиви

Символьні масиви можуть становити набір окремих символів або зв'язний символьний рядок.

У першому випадку масив можна ініціалізувати так само, як і числовий, перерахуванням всіх його елементів:

```
char cs[17]={'С','и','м','в','о','л','ь','н','и','й',' ',
             'м','а','с','и','в'};
```

Такий масив займає в пам'яті точно 16 байт.

У другому випадку масив ініціалізувався символьним рядком:

```
char cz[20] = "Символьний масив";
```

При такій ініціалізації компілятор записує в пам'ять у кінці рядка двійковий нуль, який є символом-обмежувачем. Багато функцій для роботи з рядками (наприклад, копіювання рядка або виведення рядка на екран) за цим символом визначають, де закінчується рядок. Це позбавляє нас від необхідності визначати і вказувати при виклику функції фактичну довжину рядка. Таким чином, масив займає в пам'яті на 1 байт більше, ніж в ньому є значущих символів. Цю обставину необхідно враховувати при заданні довжини масиву в його оголошенні, яка повинна вибиратися на одиницю більше максимально можливої довжини рядка.

Оголошення довжини символьного масиву із запасом доцільне лише в тих випадках, коли він заповнюватиметься програмно рядками різної довжини. Якщо ж даний текстовий рядок змінюватися не буде, зручніше оголосити його без рекомендації довжини, залишивши в оголошенні пару квадратних дужок, які скажуть компілятору, що змінна cz є не одиничним символом, а символьним масивом:

```
char cz[]="Символьний масив";
```

Компілятор, виділяючи пам'ять під цей масив, сам визначить його довжину, яка в даному випадку дорівнюватиме 17 байтам.

Приклади операцій з лінійним масивом показані у наступній програмі:

```
#include <iostream>
#include <iomanip>
#include <stdlib.h>

#define N 10
```

```
using namespace std;
int main ()
{
    int m[N];

    // 1. Ініціалізація масиву нулями.
============================
    for (int i = 0; i < N; i ++) m [i] = 0;
      // 2. Ініціалізація масиву випадковими числами
===============
    for (int i = 0; i < N; i ++) m [i] = rand()%11;
    // 3. Друк масиву в рядок.
==================================
    for (int i = 0; i < N; i ++) cout << setw(4) << m[i];
    cout << endl;
    // 4. Обчислення суми елементів масиву.
======================
    int sum = 0;
    for (int i = 0; i < N; i++) sum += m[i];
    cout << "Сума =" << sum << endl;
    // 5. Виведення елементів масиву у вигляді гістог-
рами. =======
cout << "Елемент" << setw(13) << "Значення" << setw(13)
                              << "Гістограма" << endl;
    for (int i = 0; i < N; i ++) {
       cout << setw(7) << i << setw(13) << m[i] << "..."
;
        for (int j = 0; j < m[i]; j++ ) cout << "*";
        cout << endl;
    }
    // 6. Сортування масиву методом "бульбашки".
=================
    int temp;
    cout << "Початковий порядок елементів" << endl;
   for (int i = 0; i < N; i ++) cout << setw(4) << m[i];
   cout << endl;
   // сортування
    for (int i = 0; i<N-1; i++ ) //
============================
```

```
          for (int j = i; j<N; j++ )
             if (m[i]> m[j]) {
                 temp = m[i]; m[i]= m[j]; m[j]= temp;
             }
    cout << "Масив після сортування" << endl;
for (int i = 0; i < N; i ++) cout << setw(4) << m[i];
  cout << endl;
    // 7. Лінійний пошук в масиві.
================================
    int key, flag = N;
    cout << "Введіть ключ пошуку" << endl;
    cin >> key;
    for (int i = 0; i < N; i ++) {
        if (m[i] == key ) {
            flag = i;
cout << "Шуканий елемент має індекс=" << i << endl;
           }
    }
    if (flag == N )
        cout << "Элемент не знайдений" << endl;
     // 8. Пошук максимального і мінімального елементів
масиву. ====
    int max, min;
    max = min = m[0];
    for (int i = 1; i < N; i++) {
        if (m[i] > max) max = m[i];
        if (m[i] < min) min = m[i];
    }
    cout << "Максимальний елемент=" << max << endl;
    cout << "Мінімальний елемент =" << min << endl;
    // 9. Перестановка елементів масиву в зворотному по-
рядку. =====
    for (int i = 0; i<(N/2); i++ ) {
        temp = m[i];
        m[i]=m[N-1-i];
        m[N-1-i]=temp;
    }
    // друк
```

```
    cout<< "Перестановка\n";
for (int i = 0; i < N; i ++) cout << setw(4) << m[i];
    cout << endl;
    return 0;
}
```

Якщо запустити цю програму на виконання, результат може бут таким:

```
 10  10   7   9   7   3   1   9   8   2
Сума =66
ЕлементЗначенняГістограма
      0           10...**********
      1           10...**********
      2            7...*******
      3            9...*********
      4            7...*******
      5            3...***
      6            1...*
      7            9...*********
      8            8...********
      9            2...**
Початковий порядок елементів
  10  10   7   9   7   3   1   9   8   2
Масив після сортування
   1   2   3   7   7   8   9   9  10  10
Введіть ключ пошуку
8
Шуканий елемент має індекс=5
Максимальний елемент=10
Мінімальний елемент =1
Перестановка
  10  10   9   9   8   7   7   3   2   1

Process finished with exit code 0
```

***Завдання*:** Виконати практичну частину лабораторної роботи, яка наведена у додатку А.

Контрольні питання

1. Дайте визначення масиву.
2. Як розташовується масив у пам'яті?
3. Вкажіть приклади завдання розмірності масиву.
4. Як можна ініціалізувати елементи масиву?
5. Яке інформаційне навантаження несе ім'я одновимірного масиву?
6. Яка операція використовується для визначення адреси довільного елемента масиву?

Лабораторна робота № 6
Підготовка і розв'язання на ПЕОМ задач з використанням рядків

Мета лабораторної роботи: знайомство з можливостями обробки інформації з використанням рядків, а також препроцесорної обробки даних.

Перед виконанням лабораторної роботи студент повинен знати:

- визначення і правила опису рядків з використанням засобів препроцесорної обробки;
- основні функції для роботи з рядками.

Після виконання лабораторної роботи студент повинен вміти:

- виконувати основні операції з рядками, а саме – створення, копіювання, конкатенацію, пошук підрядка в рядку та ін.

Короткі теоретичні відомості

Тип char. Значеннями типу char є цілі числа із знаком (signed char) або без знаку (unsigned char), які поміщаються в один байт. Від інших цілих типів його відрізняє наявність символічних констант вигляду:

'A' – для символів, що зображаються, '\ooo' і '\xhhh' – для всіх символів без виключення де ooo 8-річні, а hhh – 16-річні цифри.

Декілька символів мають власні імена та позначення:

`\n` – новий рядок;
`\t` – горизонтальна табуляція;
`\v` – вертикальна табуляція;
`\b` – повернення назад;
`\r` – повернення каретки;
`\a` – дзвінок (attention);
`\\` – зворотна коса межа;
`\'` – одинарна лапка;
`\"` – подвійна лапка.

Слід зауважити, що значення типу char, які виводяться у вихідний потік cout, виглядають як символи, а не як числа, тільки завдяки визначенню класу cout.

Рядки символів як масиви

Рядок має тип "масив з символів". Рядок завершується нульовим символом. Наприклад, рядок "QWERTY" має тип char[7], порожній рядок " має тип char[1].

Рядкова константа – це послідовність символів, поміщена в подвійні лапки

Рядкові константи можна використовувати для ініціалізації символьних масивів. Наприклад, так можна визначити масив s з 7 символів та ініціалізувати його:

```
char s[] = "АБВГДЕ";
```

Розглянемо приклад:

Завдання. Заданий рядок. Скопіювати її в символьний масив. Для контролю вивести в стандартний вихідний потік рядок і масив.

Розв'язання:

```
#include <iostream>
using namespace std;
int main()   {
    char s1[]="1234567890", s2[11];
    for (int i = 0; s1[i]; i++) s2[i] = s1[i];
    s2[i] = 0;
    cout << s1 << " = " << s2 << '\n';
    return 0;
}
```

Рядкові бібліотечні функції

Функції для роботи з рядками описані в заголовному файлі string.h (cstring). Деякі з них:

```
char *stpcpy(char *dest, const char *src);
```

Копіює символи рядка, поки не скопіює нульовий символ. Повертає величину dest + strlen(src). Пам'ять для dest повинна бути наперед зарезервована.

```
char *strcat(char *dest, const char *src);
```

Приєднує другий рядок до першого. Повертає покажчик на початок нарощеного рядка.

```
char *strchr(const char *s, int з);
```

Сканує рядок s в пошуку першого входження заданого символу з. Нульовий символ можна шукати разом з іншими. Повертає покажчик на знайдений символ або 0, якщо символу немає.

```
int strcmp(const char *s1, const char*s2);
```

Порівнює 2 рядки. Повертає ціле менше нуля, якщо s1 < s2, рівне нулю, якщо s1 == s2, і більше нуля, якщо s1 > s2 `char *strcpy(char *dest, const char *src);`

Копіює другий аргумент в перший. Повертає покажчик на копію. Пам'ять для dest повинна бути наперед зарезервована.

```
char *strdup(const char *s);
```

Копіює рядок у новостворювану функцією malloc() область пам'яті. Повертає покажчик на створену копію або 0 при невдачі. Програміст відповідає за звільнення пам'яті.

```
size_t strlen(const char *s);
```

Підраховує довжину рядка. Повертає кількість символів рядка без нульового символу. Тип size_t визначений у файлі string.h і інших заголовних файлах як ціле без знаку: typedef unsigned size_t;

```
char *strpbrk (const char *s1, const char *s2);
```

Сканує перший рядок у пошуках першого входження будь-якого символу з другого рядка. Повертає покажчик на знайдений символ або 0 при невдачі

```
char *strrchr (const char *s, int a);
```

Те ж, що strchr, але знаходить останнє входження символу a в рядок s.

```
char *strset(char *s, int ch);
```

Пише символ ch замість усіх символів рядка. Повертає s.

```
char *strstr(const char *s1, const char *s2);
```

Знаходить перше входження підрядка s1 у рядок s1. Повертає покажчик на місце першого входження або 0, якщо такого немає.

```
char *strtok(char *s1, const char *s2);
```

Сканує перший рядок у пошуках першої ділянки, що не містить символів з s2. Перший виклик функції повертає покажчик на початок першої

ділянки і записує 0 в s1 відразу після кінця ділянки. Подальші виклики з NULL як 1-й аргумент обробляють рядок далі, поки що є такі ділянки. Якщо їх немає, повертається 0. Рядок s2 може змінюватися від виклику до виклику.

Функцію застосовують для виділення слів з пропозиції s1. У рядку s2 знаходяться символи-роздільники.

***Завдання*:** Виконати практичну частину лабораторної роботи, яка наведена у додатку А.

Контрольні питання

1. Дайте визначення рядка.
2. Що представляє собою рядок у мові С++?
3. Як визначити довжину рядка?
4. Який символ використовується як ознака кінця рядка?
5. Як відбувається введення-виведення рядкових величин?
6. У якому заголовковому файлі знаходиться опис рядкових функцій?

Додаток

Завдання лабораторних робіт по варіантах

Лабораторна робота № 1

Завдання 1.1. Записати мовою С++ представлені математичні вирази.

№ варіанта	Завдання	№ варіанта	Завдання
1–3	$a)\dfrac{(\ln\lvert 2z\rvert+\operatorname{arctg} 2z^2)}{3(z+1)^2+2{,}1\cdot 10^6}$ $b)\ln\lvert x+z\rvert>0$ та $0<b<1$	4–6	$a)\dfrac{(\ln\lvert 5z\rvert+\operatorname{arctg}^2 3)}{3(z+1)^2+2{,}1\cdot 10^{-6}}$ $b)\ln\lvert x+z\rvert>0$ та $0<b<1$
7–9	$a)\dfrac{(10^{-7}\ln\lvert 2z\rvert+\sin 2z^3)}{3(z+3)^2+2{,}1\cdot 10^7}$ $b)\lvert x+z\rvert>1$ та $1<b<2$	10–12	$a)\dfrac{(10^{-7}\ln\lvert 2z\rvert+b^{0{,}4})}{\ln(z+1)^2+4{,}2\cdot 10^4}$ $b)\lvert x\rvert>2$ або $0<b<3$
13–15	$a)\dfrac{(10^{-5}e^{-5f}+\sin^2\lvert z^3\rvert)}{5(z+1)^5+10^6}$ $b)x+z<0$ або $0<f<0{,}2$	16–18	$a)\dfrac{(10^{-4}e^{-2f}+\ln\lvert z^3\rvert)}{2(z+2)^{1{,}5}}$ $b)\cos\lvert x+z\rvert>0$ або $0<b<3$
19–21	$a)\dfrac{(\ln\lvert 3z\rvert+\operatorname{arctg} 2z^2)}{3(z+1)^2+2{,}1\cdot 10^6}$ $b)\lvert x+z\rvert>0$ та $0<b<7$	22–24	$a)\dfrac{(10^{-7}\sin\lvert 3z\rvert+b^{1{,}2})}{(z+1)^2+1{,}2\cdot 10^6}$ $b)\ln\lvert x+z\rvert>0$ або $0<b<1$
25–27	$a)\dfrac{(10^{-6}\ln\lvert z^3\rvert+\ln^2 z^3)}{6(z+1)^6+10^6}$ $b)\cos\lvert x+z\rvert>0$ або $0<b<3$	28–30	$a)\dfrac{(10^{-7}\ln\lvert 3z^3\rvert+\sin 2z^2)}{(z+1)^{0{,}5}+10^6}$ $b)\lvert x+z\rvert>0$ або $0<b<1$

Продовж. завдання 1.1

№ варіанта	Завдання	№ варіанта	Завдання
31–33	$a)\dfrac{(\ln\lvert 4z\rvert+\operatorname{arctg}^3 2z)}{4(z+1)^{0,2}+1,7\cdot 10^3}$ $b)\lvert x+z\rvert>0$ або $0<b<7$	34–36	$a)\dfrac{(10^{-5}e^{-3f}+\ln\lvert z^{-3}\rvert)}{5(z+2)^{2,5}}$ $b)x+z<0$ або $0<f<0,2$

Завдання 1.2. Представити математичний запис виразу, записаного мовою С++, і показати порядок дій.

№ варіанта	Завдання
1, 19	`x+2.0/3.0/x/a+sqrt(sin(x))/2*sqrt(x)+1.0e-6* pow(x,1.0/7.0)`
2, 20	`(x+7)/3*x+3*atan(x)/2/x+1.0e7-sqrt(1.0/3.0*pow(x,5))`
3, 21	`x+2/3.0*x/a+sqrt(cos(x))/2/sqrt(x)+1.0e-5*pow(x,7)`
4, 22	`(x+4)/3/x+exp(abs(atan(x)))/2*x+1.0e-6*pow(x,1.0/3)`
5, 23	`x+2/3.0/x/a+sqrt(sin(x))/2.0/ln(x)+1.0e5*pow(x/3,2/7.0)`
6, 24	`1.4e-4*pow(2*x,3)+sqrt(sin(x))/2+sqrt(cos(x))/2/x`
7, 25	`sqrt(cos(x))/2/x-5.0/7.0*x/a/1.0e-6* pow(x/2,1/3.0)*abs(x)`
8, 26	`x+2.0/3/x*a+sqrt(sin(x))/2/ln(x)+1.0e-3*pow(x/7,2.0/3)`
9, 27	`(x+7)/3*x+3*atan(x)/2/x+1.0e7-sqrt(4*pow(x,b))`
10, 28	`sqrt(cos(abs(x)))/2/x-5.0/7*x/a/1.0e-6*pow(x/2,1.0/8.0)`
11, 29	`x+9/(3*x/A)+sqrt(cos(x))/2/sqrt(x)+1.0e-5*pow(x,9)`
12, 30	`x+4/3.0/x+exp(abs(atan(x)))/2*x+1.0e-4*pow(x,1.0/3.0)`
13, 31	`sqrt(abs(cos(x)))/2/(x-5.0/7)*x/a/1.0e-6*pow(x/2,5/3.0)`
14, 32	`x+2*3/x*a+ln(abs(sin(x)))/(2*cos(x)+1.0e-3* pow(x/2,1.0/7))`
15, 33	`x+4.0/3/(x+abs(atan(x)))/2*x+1.0e-5*pow(x,5.0/3.0)`
16, 34	`sqrt(cos(x))/2*x-4.0/3*x/a/1.0e8*pow(x/3,2.0/3.0)* sin(x)`
17, 35	`ln(x+5)/2*x+4*atan(x)/5/x+1.0e5 -sqrt(4*pow(x,b/(2.0/3.0)))`
18, 36	`x+5.0/(3*x/A)+ln(abs(cos(x)))/2/exp(x)+1.0e-5*pow(x,3)`

Завдання 1.3. Скласти програму обчислення наступних величин та виконати її в інтегрованому середовищі розробки (IDE). Позначення: N – номер варіанта за списком групи.

№	Умова
1	Модуль вектора 5**a**+10**b**, якщо **a**={3; 2} і **b**={0; -1}
2–6	Сума всіх парних чисел від 2 до 50***N**
7–11	Сума всіх двозначних цілих чисел, які кратні **N**
12	Кут між векторами **a**={1; 2} і **b**={1; –0,5}
13	Площа чотирикутника з вершинами A(0; 0), B(–1; 3), C(2; 4), D(3; 1)
14	Сума одинадцяти перших членів арифметичної прогресії, якщо $a_3+ a_9=8$
15	Периметр трикутника з вершинами A(1; 1), B(4; 1), C(4; 5)
16	Модуль вектора -2**a** + 4**b**, якщо **a**={3; 2}, **b**={0; -1}
17	Кути трикутника з вершинами A(0; 1,7), B(2; 1,7), C(1,5; 0,85)
18	Шостий член геометричної прогресії 5, -10, ...
19	Кут між векторами a={2; -4; 4} та b={-3; 2; 6}
20	Модуль вектора **a**-**b**, якщо |**a**|=3, |**b**|=5 та ці вектори утворюють кут у 120º
21	Сума усіх двозначних цілих чисел
22	Модуль вектора **a**+**b**, якщо |**a**|=11, |**b**|=23, |**a**-**b**|=30
23	Сума усіх непарних двозначних чисел
24–28	Сума усіх тризначних цілих чисел, які у разі ділення на 5 дають остачу 28-**N**
29–32	Сума усіх непарних чисел від 3 до 5***N**
33–36	Сума усіх парних чисел від 10 до 7***N**

Лабораторна робота № 2

Завдання 2.1. Представити математичний запис фрагменту програми та обчислити значення змінної X після його виконання. Позначення: N – це номер варіанта.

№ варіанта	Фрагмент програми	№ варіанта	Фрагмент програми

1–2	`t=17*n; x=t;`
	`if (t<10 || t>30) x=3;`
	`  else if (t<=20) x=0;`
		3–4	`t=n; x=0;`
			`if (t<0)x= -t;`
			`  else x= t;`

Продовж. завдання 2.1

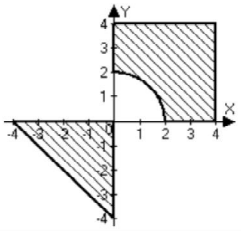
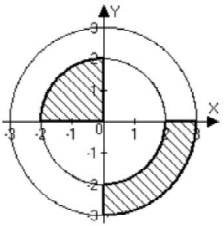
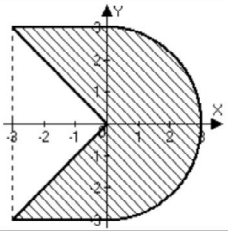
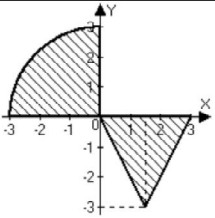
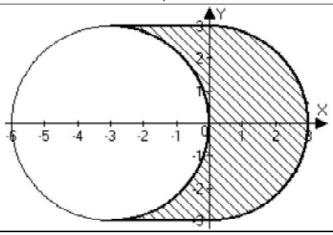
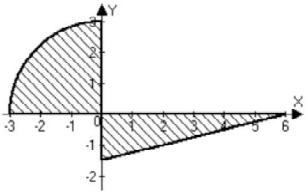
№ варіанта	Фрагмент програми	№ варіанта	Фрагмент програми

5–6	a=n; b=13; c=12; x=a;
	if (x<b) x=b;
	if (x<c) x=c;
		7–8	a=n; b=17; c=18; x=a;
			if (b<x) x=b;
			if (c<x) x=c;

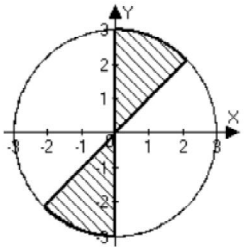
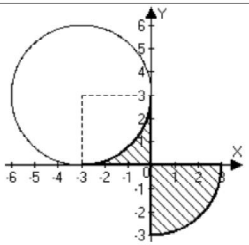
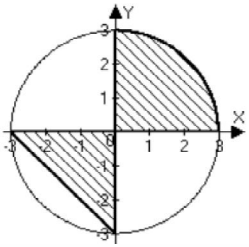
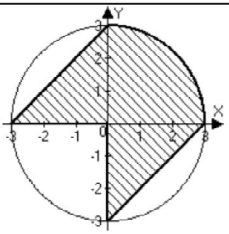
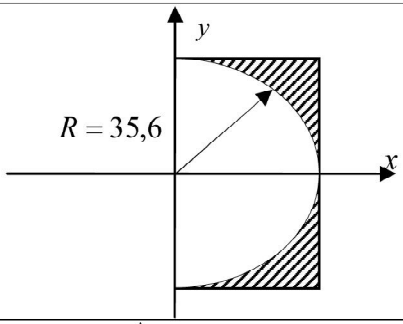
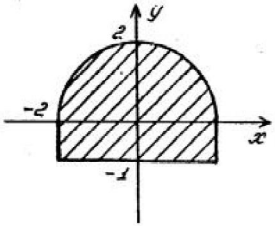
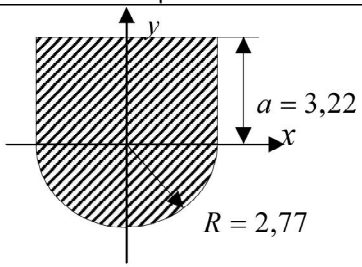
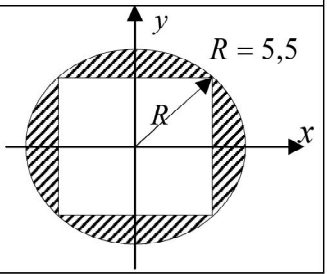
9–10	t=n; x=0;
	if (t>10) x=t*t-n;
	if (t<10) x=t;
		11–12	t=n; x=t%4;
			if(t>1 && t<3) x=t;
			if (t<=1) x=1;

13–14	t=n; x=t;
	if(t>0 && t<10) x=1;
	if (t>=10) x=1/(exp(t)-1);
		15–16	x=-7;t=pow(x,n);
			if (t>0)x=pow(t,1.0/3);
			else x= t*t*t;

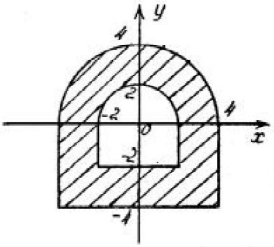
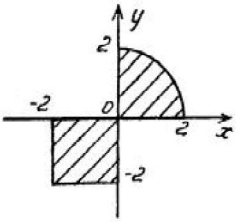
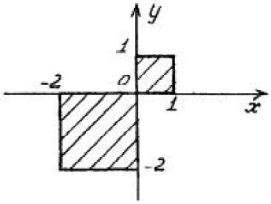
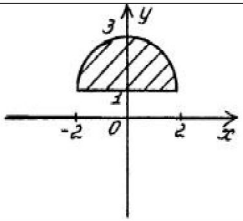
Завдання 2.2. Написати програму, яка виводить на екран значення true, якщо точка A з координатами x, y належить області, що заштрихована, та false в іншому випадку.

№ варіанта	Область	№ варіанта	Область
1		2	
3		4	
5		6	

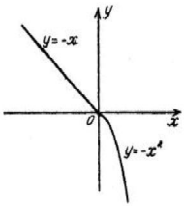
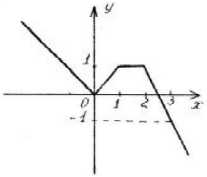
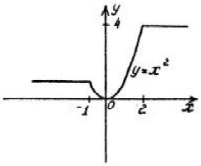
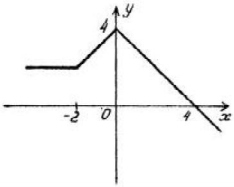
Продовж. завдання 2.2

№ варі-анта	Область	№ варі-анта	Область
7		8	
9		10	
11		12	
13		14	

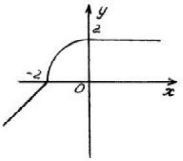
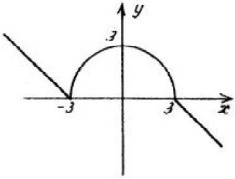
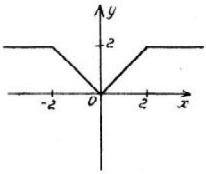
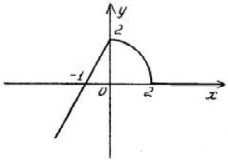
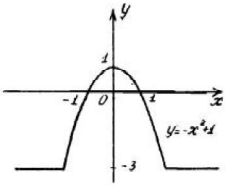
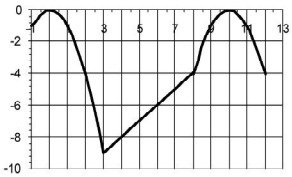
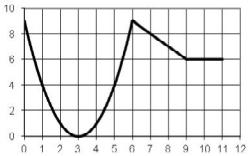
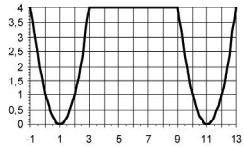
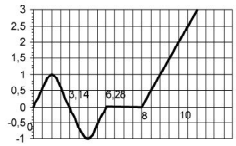
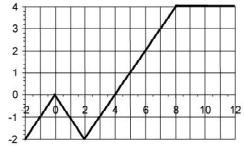
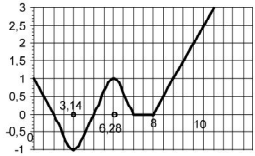
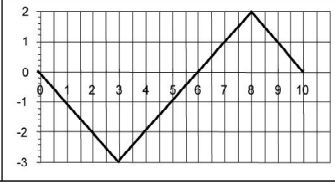
Продовж. завдання 2.2

№ варіанта	Область	№ варіанта	Область
15		16	
17		18	

Завдання 2.3. Скласти програму обчислення значення функції, яка задана графічно. (Значення аргументу X вводиться довільно з клавіатури).

№ варіанта	Функція	№ варіанта	Функція
1		2	
3		4	

Продовж. завдання 2.3

№ варіанта	Функція	№ варіанта	Функція
5		6	
7		8	
9		10	
11		12	
13		14	
15		16	

Лабораторна робота № 3

Завдання 3.1. Представити математичний запис фрагмента програми та обчислити значення змінної x після його виконання, де N – це номер варіанта.

№ варіанта	Завдання	№ варіанта	Завдання

1–3	`x = 1;`
	`for (int j=7; j>=N; j--`
	`  x *= j;`
	`x *= 2;`
		4–6	`x = 0; j = 1;`
			`do {`
			`  x += j; j += 2;`
			`while (j<=N);`

7–9	`x = 0; k = 3 * N;`
	`while (k > 0) {`
	`  x = sqrt(k+x);`
	`  k -= 3;`
	`}`
		10–12	`x = N;`
			`for (int k=0; k<6;`
			`k++) {`
			`  if (k<2) continue;`
			`  x++;`
			`}`

13–15	`x = 0;`
	`for (int j=0; j<N; j++)`
	`  x += 2;`
	`x *= 2;`
		16–18	`x = 1;`
			`while (x<=N)`
			`  x++;`
			`x *= 2;`

19–21	`x = 1.0 / N; n = N;`
	`while (n>1) {`
	`  n -= 2;`
	`  x = 1.0 / (n + x);`
	`}`
		22–25	`x = 1.0/N; n = N;k = 1;`
			`while (n>1) {`
			`  n -=2; k = -k;`
			`  x = 1.0/(n + k * x);`
			`}`

Завдання 3.2. Скласти програму табулювання функції $y = f(x)$ на відрізку $[a; b]$ з кроком h. Значення a, b, h вводити з клавіатури.

№ варіанта	Функція	№ варіанта	Функція
1	$y = \frac{\operatorname{tg} x}{\ln x}$	7	$y = x^{0,2}$
2	$y = \sqrt[3]{x}$	8	$y = \frac{x}{1 + \operatorname{tg} x}$
3	$y = \operatorname{tg}(\ln x)$	9	$y = x^{\frac{1}{7}}$
4	$y = \frac{\ln(x-1)}{4-x}$	10	$y = \ln(\operatorname{tg} x)$
5	$y = \operatorname{tg}^2(\ln x)$	11	$y = \frac{x^3}{\cos x}$
6	$y = \operatorname{ctg}(\ln x)$	12	$y = \frac{\sqrt{\lvert x \rvert}}{\sin x}$

Продовж. завдання 3.2

№ варіанта	Функція	№ варіанта	Функція
13	$y=\sqrt{\frac{1}{x^3}}$	20	$y=\frac{\ln\lvert 2x\rvert}{\sin x-\pi}$
14	$y=\frac{\ln(x-0.5)}{\sqrt{x}}$	21	$y=e^{\ln x+1}+\sin x$
15	$y=\frac{\sin x^3}{2-x}$	22	$y=\frac{\operatorname{ctg} x}{x^x-1}$
16	$y=\frac{\cos^3 x}{1-\lg x}$	23	$y=\frac{x^{15}}{1-x}$
17	$y=\frac{\operatorname{tg} x}{\ln x-1}$	24	$y=\ln x-\frac{1}{x^2}$
18	$y=\frac{e^2}{\sqrt{1-x^2}}$	25	$y=\frac{1}{x^2}+\frac{\sin x}{x}$
19	$y=\frac{x+1}{\sqrt{x}}-\sqrt[4]{\lvert x-2\rvert}$		

Завдання 3.3. Для заданих x, n, e, що вводяться з клавіатури:

- обчислити n доданків згідно з варіантом;
- обчислити суму тих доданків, які за абсолютним значенням більше e. (Завдання виконати для двох різних e, які відрізняються на порядок, для кожного випадку обчислити кількість доданків);
- порівняти результати з точним значенням відповідної функції (сума визначає наближене значення) для x ∈ (-R,R).

№ варіанта	Функція
1	$\frac{\sin x}{x}=1-\frac{x^2}{3!}+\frac{x^4}{5!}-\frac{x^6}{7!}+\ldots \quad (R=\infty)$
2	$e^{-x^2}=1-\frac{x^2}{3!}+\frac{x^4}{5!}-\ldots \quad (R=\infty)$
3	$\ln\left(x+\sqrt{x^2+1}\right)=x-\frac{1}{2}\cdot\frac{x^3}{3}+\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{x^5}{5}-\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot\frac{x^7}{7}+\ldots \quad (R=1)$
4	$\operatorname{arctg} x=x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+\frac{x^9}{9}-\ldots \quad (R=1)$
5	$\arcsin x=x+\frac{1}{2}\cdot\frac{x^3}{3}+\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{x^5}{5}+\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot\frac{x^7}{7}+\ldots \quad (R=1)$

Продовж. завдання 3.3

№ варіанта	Функція
6	$\frac{1}{\sqrt{1-x^2}} = 1 + \frac{1}{2}\cdot x^2 + \frac{1}{2}\cdot\frac{3}{4}\cdot x^4 + \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot x^6 + \ldots \quad (R=1)$
7	$\frac{1}{\sqrt{1-x}} = 1 - \frac{1}{2}\cdot x + \frac{1}{2}\cdot\frac{3}{4}\cdot x^2 + \frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot x^3 + \ldots \quad (R=1)$
8	$\sqrt{1-x} = 1 + \frac{1}{2}\cdot x - \frac{1}{2\cdot 4}\cdot x^2 + \frac{1\cdot 3}{2\cdot 4\cdot 6}\cdot x^3 - \ldots \quad (R=1)$
9	$\frac{1}{(1+x)^2} = 1 - \frac{2\cdot 3}{2}\cdot x + \frac{3\cdot 4}{2}\cdot x^2 - \frac{4\cdot 5}{2}\cdot x^3 + \ldots \quad (R=1)$
10	$\frac{1}{(1+x)^2} = 1 - 2x + 3x^2 - 4x^3 + 5x^4 - \ldots \quad (R=1)$
11	$\frac{1}{1+x} = 1 - x + x^2 - x^3 + x^4 - \ldots \quad (R=1)$
12	$\frac{\ln(1+x)}{1-x} = 2\cdot\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \frac{x^9}{9} + \ldots\right) \quad (R=1)$
13	$\ln(1+x) = -\frac{x}{1} - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \ldots \quad (R=1)$
14	$\ln(1+x) = \frac{x}{1} - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots \quad (R=1)$
15	$ch(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots \quad (R=\infty)$
16	$sh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \ldots \quad (R=\infty)$
17	$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots \quad (R=\infty)$
18	$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \quad (R=\infty)$
19	$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots \quad (R=\infty)$
20	$e^{-x} = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \ldots \quad (R=\infty)$

Лабораторна робота № 4

Завдання 4.1. Представити математичний запис фрагмента програми та обчислити значення змінної x після його виконання. Елементи масиву обчислюються за формулою `a[i+1] = (67 * a[i] + 11) % 128`. Значення a[0] дорівнює номеру варіанта за списком групи.

№ варіанта	Фрагмент програми	№ варіанта	Фрагмент програми

1–5	`t=2; n=3; x = a[0];`
	`for(int j=0; j<n;`
	`j++){`
	`  x=x*t+a[j+1];`
	`}`
		6–10	`n=4; x = a[n];`
			`for (int j=n-1; j>=0;`
			`j--){`
			`  x=a[j]+a/x;`
			`}`

11–15	`n=4; x = a[0];`
	`for(int j=1; j<n;`
	`j++){`
	`  if (a[j]<x) x=a[j];`
	`}`
		16–20	`t=3; n=3; x=a[n];`
			`for (int j=0; j<n; j++){`
			`  x=x+a[j]*pow(t,n-j);`
			`}`

21–25	`n=4; m=n/2; k=n-1;`
	`for(int j=0; j<m;`
	`j++){`
	`  y=a[j];`
	`a[j]=a[k];`
	`  a[k]=y; k--;`
	`}`
	`x=a[0];`
		26–30	`n=4; k=0; x=0;`
			`for (int j=0; j<n; j++)`
			`  if(a[j]>0){`
			`  x+=a[j]; k++;`
			`  }`
			`if (k!=0) x/=k;`

Завдання 4.2. Скласти програму обчислення наступних величин та виконати її у середовищі програмування. Елементи масиву визначаються за формулою `a[i] = p[i] - 64; де p[i+1]=(p[i]*67 + 11)%128;` p[0] дорівнює N – номеру варіанта за списком групи, кількість елементів у масиві дорівнює 50.

№ варіанта	Умова
1–3	Найбільший елемент масиву a та його порядковий номер
4–6	Сума елементів масиву a, значення яких кратні N
7–9	Сума елементів масиву a, значення яких парні числа
10–12	Середнє арифметичне додатних елементів масиву a
13–15	Сума елементів масиву a, значення яких непарні числа
16–18	Середнє геометричне додатних елементів масиву a
19–21	Сума елементів масиву a, значення яких двозначні непарні числа
22–24	Добуток найбільшого та найменшого елементів масиву a
25–27	Сума елементів масиву a, значення яких двозначні парні числа
28–30	Модуль вектора a/3

Завдання 4.3. Протабулювати функцію із завдання 3.2. Значення x та y занести у масиви. Знайти найбільше та найменше значення у масиві y. Вивести їх та відповідні їм значення з масиву x у наступному вигляді:

```
yMin = … при x = …
yMax = … при x = …
```

Обчислити суму та середнє арифметичне значення елементів масиву y. Результати вивести на екран.

Лабораторна робота № 5

Завдання 5.1. Обчислити значення змінних x, y після виконання фрагмента програми. (N – номер варіанта за списком групи)

№ варіанта	Фрагмент програми	№ варіанта	Фрагмент програми

1–5	`#include <iostream>`
	`using namespace std;`
	`double a, x, y;`
	`void d(double &x, &y) {`
	`  x=2*x; y=a*x+1;`
	`}`
	`int main() {`
	`  a = N; x = 3; y = 4;`
	`  d(y, x); y = a*x;`
	`  cout << x << " " <<`
	`  y << endl;`
	`}`
		6–10	`#include <iostream>`
			`using namespace std;`
			`double a, x, y;`
			`void d(double &x, &y) {`
			`  x=2*x; y=a*x+1;`
			`}`
			`int main() {`
			`  a = N-5; x = 3; y = 4;`
			`  d(a, x); y = a*x;`
			`  cout << x << " " <<`
			`  y << endl;`
			`}`

11–15	`#include <iostream>`
	`using namespace std;`
	`double a, x, y;`
	`void d(double &x, &y) {`
	`  x=2*x; y=a*x+1;`
	`}`
	`int main() {`
	`  a = N-5; x = 2; y = 3;`
	`  d(a, x); y = a*x;`
	`  cout << x << " " <<`
	`  y << endl;`
	`}`
		16–20	`#include <iostream>`
			`using namespace std;`
			`double a, x, y;`
			`void d(double &x, &y) {`
			`  x=2*x; y=a*x+1;`
			`}`
			`int main() {`
			`  a = N-10; x = 3; y = 2;`
			`  d(a, y); y = a*x+y;`
			`  cout << x << " " <<`
			`  y << endl;`
			`}`

№ варіанта	Фрагмент програми	№ варіанта	Фрагмент програми

21–25	`#include <iostream>`
	`using namespace std;`
	`double x, y;`
	`double f(double x) {`
	`  if (x==0) return 1;`
	`  else return 2*x*f(x-1);`
	`}`
	`int main() {`
	`  x = N-18; y = f(x);`
	`  cout << x << " " <<`
	`  y << endl;`
	`}`
		26–30	`#include <iostream>`
			`using namespace std;`
			`double x, y;`
			`double f(double x) {`
			`  if (x==0) return 0;`
			`  else return x + f(x-1);`
			`}`
			`int main() {`
			`  x = N-23; y = f(x);`
			`  cout << x << " " <<`
			`  y << endl;`
			`}`

Завдання 5.2. Скласти програму згідно з завданням 4.3 і виконати її у середовищі програмування. При розробці програми для виконання завдань згідно з пунктами (b) та (c) використовувати функції.

Лабораторна робота № 6

Завдання 6.1. Скласти програму для виконання наступних дій і виконати її в середовищі програмування.

№ варіанта	Завдання
1–3	Визначення кількості слів у рядку
4–6	Вилучення всіх цифр у рядку
7–9	Інвертування символів у рядку
10–12	Визначення кількості цифр у рядку
13–15	Визначення слова з найменшою кількістю літер у рядку
16–18	Визначення кількості чисел у рядку
19–21	Визначення слова з найбільшою кількістю літер у рядку
22–24	Заміна усіх великих букв у рядку на малі
25–27	Вилучення зайвих символів «пробіл» у рядку
28–30	Заміна усіх малих букв у рядку на малі

Завдання 6.2. Скласти програму для виконання наступних дій і виконати її в середовищі програмування.

№ варіанта	Завдання
1–2	У тексті знайти перший підрядок максимальної довжини, що не містить букв
3–4	У тексті визначити всі приголосні букви, що зустрічаються не більше, ніж у двох словах

Продовж. завдання 6.2

№ варіанта	Завдання
5–6	Перетворити текст так, щоб кожне слово, яке не містить неалфавітних символів, починалося з великої літери.
7–8	Підрахувати кількість розділових знаків які містяться в даному тексті
9–10	У заданому тексті знайти суму всіх цифр, які зустрічаються
11–12	Всі слова тексту зустрічаються парну кількість разів, за винятком одного. Визначити це слово
13–14	Визначити суму всіх цілих чисел, що зустрічаються в заданому тексті
15–16	З тексту видалити всі зайві пробіли, якщо вони розділяють два різних розділових знаки і якщо поряд з ними знаходиться ще один пробіл
17–18	Рядок складається з упорядкованих чисел від 0 до 100 000, записаних поспіль без пробілів. Визначити, що буде підрядком від позиції n до m
19–20	Визначити кількість входжень заданого слова в текст, ігноруючи регістр символів
21–22	Перетворити текст так, щоб тільки перші літери кожного речення були великими
23–24	Замінити всі однакові символи, які стоять поруч, одним символом
25–26	Вивести в заданому тексті всі слова, розташувавши їх в алфавітному порядку
27–28	Підрахувати, скільки слів у заданому тексті починається з великої літери
29–30	Підрахувати, скільки разів задане слово входить у текст

Зміст