ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КЕРУЮЧИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Кафедра обчислювальної техніки та систем управління

ОСНОВИ ПРОГРАМУВАННЯ МОВОЮ С++

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисциплін

“*ПРОГРАМУВАННЯ*", "*ІНФОРМАТИКА*", "*АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ*"

Частина 1

Харків – 2017

Методичні вказівки розглянуто і рекомендовано до друку на засіданні кафедри обчислювальної техніки та системи управління 27 березня 2017 р., протокол № 8.

Призначено для студентів факультету ІКСТ денної та заочної форм навчання.

Укладачі:

доценти С. Є. Бантюков,
В. М. Бутенко,
О. В. Головко,
С. О. Бантюкова,
старш. викл. О. В. Чаленко

Рецензент

проф. С. В. Лістровий

ОСНОВИ ПРОГРАМУВАННЯ МОВОЮ С++

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт з дисциплін
“*ПРОГРАМУВАННЯ*", "*ІНФОРМАТИКА*",
"*АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ*"

Частина 1

Відповідальний за випуск Головко О. В.

Редактор Решетилова В. В.

Підписано до друку 30.03.17 р.

Формат паперу 60х84 1/16. Папір писальний.

Умовн.-друк.арк. 2,50. Тираж 50. Замовлення №

Видавець та виготовлювач Українська державна академія залізничного транспорту,
61050, Харків-50, майдан Фейєрбаха, 7.
Свідоцтво суб’єкта видавничої справи ДК № 2874 від 12.06.2007 р.

ВСТУП

У теперішній час сфера розповсюдження обчислювальної техніки не має кордонів. Персональні ЕОМ дозволяють вирішувати важливі виробничі завдання, вести навчання фахівців, управляти складними процесами і т.п. Зараз вже очевидно, що навички роботи з такою технікою необхідні кожному інженерові.

Сьогодні ЕОМ виступає у ролі приємного співрозмовника, не стомлює, підказує, нагадує. Діалог між ЕОМ та людиною ведеться мовою, що допускає його однозначне тулмачення з обох сторін. Важко перерахувати всі мови та інструментальні системи програмування для ЕОМ, що реалізують їх. Найбільш популярними та широко використовуваними є Basic, C, Pascal.

Ці методичні вказівки присвячені мові програмування C++. Вони складаються з восьми робіт, які містять опис роботи у середовищі програмування Borland C++, версії 3.1 (далі по тексту C++), елементарні конструкції мови C++, основні оператори та функції для програмування лінійних, розгалужених, циклічних обчислювальних процесів, обробки масивів.

C++ вважається мовою середнього рівня, тому що вона об'єднує у собі елементи мов високого рівня з функціональністю Асемблера.

Програма, написана мовою C++, дуже мобільна. Це означає, що програмне забезпечення, написане для комп'ютера одного типу, можна легко адаптувати для комп'ютера іншого типу.

Мову C++ часто називають структурованою мовою. Відмітною рисою структурованої мови є наявність можливості відокремлювати дані від програми та розбивати програму на окремі функції, кожна з яких є незалежною одиницею, що виконує певну задачу та допускає виключеня її з програми.

При використанні системи Borland C++ користувачі мають у своєму розпорядженні найшвидший та ефективний за можливостями, компілятор та інтегроване середовище, що дозволяє максимально автоматизувати процес розроблення програм.

Завдання до лабораторних робіт знаходяться в «МЕТОДИЧНИХ ВКАЗІВКАХ до лабораторних робіт 1-9 з дисципліни «ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ». Частина 1» в кінці відповідних робіт.

В методичних вказівках в записах форматів застосовуються такі позначення:

<*значення*> - обов'язковий елемент, на місці якого повинна бути послідовність символів, що допущена у даному форматі;

[рядок_символів] – рядок сиволів, взятих у квадратні дужки, вважається необов'язковим і може бути відсутнім.

Лабораторна робота 1. Робота в інтегрованому середовищі програмування Borland C++ версії 3.1

1 Мета роботи: отримання навичок роботи в інтегрованому середовищі програмування Borland C++ версії 3.1, ознайомлення з командами основного меню, основними командами редактора.

2 Завдання та порядок виконання

2.1 Вивчити теоретичний матеріал.

2.2 Підготувати відповіді на контрольні запитання.

2.3 Запустити Borland C++. У вікні редактора набрати приклад навчальної програми. Випробувати команди редактора C++. Випробувати команди основного меню.

3 Контрольні запитання

3.1 Чому середовище для розробки C++-програм називається інтегрованим?

3.2 З яких частин складається екран основного меню? Їх призначення.

3.3 Призначення опцій основного меню.

3.4 Команди редактора C++.

3.5 Команди основного меню.

4 Зміст звіту

4.1 Номер роботи, її назва, визначення мети.

4.2 Стислий зміст теоретичного матеріалу та відповіді на контрольні запитання.

4.3 Результати виконання завдання: схеми алгоритмів, програми, результати роботи програм.

4.4 Висновки до роботи.

5 Навчальний матеріал

5.1 Запуск C++. Для запуску C++ слід набрати у командному рядку *bc* і натиснути клавішу *Enter* або обрати опцію *«Borland C++»* (опція - від англ. *option* - вибір) в меню користувача, що викликається натисканням функціональної клавіші *F2*. При цьому на екрані з'явиться картинка, що називається екраном основного меню і складається з чотирьох частин:

- основного меню;
- вікна редактора;
- вікна повідомлень;
- рядка підказки чи вказівки дій функціональних клавіш.

Для виходу з середовища С++ використовується комбінація клавіш *Alt+X* або команда *Quit* (Вихід) з опції *File* основного меню.

Розглянемо кожну з чотирьох частин.

Основне меню. Основне меню використовується або для вказування дій, або для встановлення певних опцій середовища. Для переходу в основне меню використовують функціональну клавішу *F10*. Для вибору команд з основного меню користуються клавішами управління курсором, переміщаючи підсвітку на необхідну опцію та натискаючи клавищу введення. В таблиці 1.1 описуються дії кожної опції.

Таблиця 1.1 – Дія опцій основного меню

Опція	**Дія**
≡	Системні команди та програми перетворення
File	Команди управління файлами (завантаження, збереження і т.п.), виклик команд ОС, вихід з С++
Edit	Містить команди редактора С++
Search	Виконує пошук, заміну символів у тексті, перехід до заданого рядка
Run	Компілює та виконує програму
Compile	Компілює програму, управляє багатофайловими об'єктами
Debug	Дозволяє влаштовувати різноманітні параметри налагодження
Project	Використовується для створення та супроводу великих багатофайлових програм
Options	Дозволяє влаштовувати різноманітні параметри оточення, компілятора та компоновача
Window	Використовується при роботі з вікнами
Help	Допомога

Вікно редагування. Під головним меню знаходиться вікно редагування. Програма редагується всередині цього вікна.

Вікно повідомлень. Вікно повідомлень знаходиться під вікном редагування та використовується для відображення повідомлень компілятора та компоновача.

Рядок підказки чи вказування дій функціональних клавіш. При використанні основного меню в цьому рядку показується підказка про те, що виконує та чи інша команда. У режимі редагування в цьому рядку показуються функціональні клавіші інтерактивного середовища С++, що використовуються для активізації меню чи швидкого виконання стандартних функцій. Так як функціональні клавіші постійно активні, з їхньою допомогою можна викликати будь-яке меню незважаючи на поточну роботу. В рядку показуються функціональні клавіші, найбільш придатні для роботи, що виконується у даний момент.

5.2 Використання редактора С++. Редактор С++ дозволяє маніпулювати цілими блоками тексту. С++-дії вимагають, щоб спочатку була виділена необхідна частина тексту, а потім виконана дія. Зробити виділення можна, помістивши курсор у початок блока, що виділяється, і натиснути комбінацію *Shift+клавіші управління курсором (стрілки)*. Курсор необхідно пересунути у кінець ділянки, що виділяється. При цьому С++ висвітить відмічену ділянку тексту.

Для переміщування блока необхідно спочатку забрати виділений блок у пам'ять - комбінація клавіш *Shift+Del* (при цьому виділений блок зникне на екрані), перемістити курсор у нове місце призначення та натиснути *Shift+Ins*. Блок з'явиться в новому місці. В пам'яті може зберігатися тільки один виділений блок. При наступному запам'ятовуваанні блока попередній блок буде знищено.

Для копіювання виділеного блока спочатку необхідно натиснути *Ctrl+Ins*, перемістити курсор у нове місце призначення та натиснути *Shift+Ins*. В новому місці з'явиться копія виділеного блока.

Для вилучення виділеного блока необхідно натиснути *Ctrl+Del*.

Для скасування останньої виконаної команди необхідно натиснути *Alt+Backspace*.

5.3 Використання команд основного меню. Нижче наведена таблиця 1.2 деяких команд основного меню та відповідних їм клавіш для виконання основних дій при налагодженні та виконанні програм.

Таблиця 1.2 = Команди та відповідні клавіші основного меню

Клавіша	**Команда**
Shift+F1	Виклик допомоги
Ctrl+F1	Виклик контекстнозалежної допомоги
F2	Збереження файлу
F3	Відкриття файлу
F9	Компілювання програми
Ctrl+F9	Виконання програми
F5	Розкрити вікно / стиснути вікно
F6	Наступне вікно
Alt+F3	Закрити вікно
Alt+F5	Екран користувача

Після набору тексту програми у вікні редактора програму необхідно виконати. Для цього використовується комбінація клавіш *Ctrl+F9*. Після натискання *Ctrl+F9* на екрані з'явиться вікно компілятора. В нижньому рядку вікна компілятора виводяться повідомлення про результат компіляції. Якщо програма не містить синтаксичних помилок, в нижньому рядку з'являється повідомлення «Success : Press any key» і на екрані користувача відображається результат виконання програми. Для виклика екрана користувача необхідно натиснути комбінацію клавіш *Alt+F5*, при цьому екран основного меню зникне. Для повернення екрану основного меню необхідно натиснути будь-яку клавішу. Якщо в програмі містяться синтаксичні помилки, то у нижньому рядку з'являється повідомлення «Errors: Press any key».

Після натискання будь-якої клавіші вікно компілятора зникне і активним стане вікно повідомлень, у якому з'являться повідомлення про найбільш імовірні помилки. Якщо у вікні повідомлень містяться повідомлення про деякі помилки, то при пересуванні курсором по рядках повідомлень про помилки у вікні редактора курсор буде синхронно пересуватися по рядках, які містять зазначену помилку.

Для переходу у вікно редагування для виправлення помилок необхідно натиснути клавішу *F6*. Після виправлення помилок програму необхідно знову виконати, натиснувши комбінацію клавіш *Ctrl+F9*. Слід мати на увазі, що одна помилка може потягти за собою кілька інших помилок, усунувши яку, усуваються й інші.

Лабораторна робота 2. Вивчення елементарних конструкцій мови С++

1 Мета роботи: вивчення правил запису елементарних конструкцій мови програмування С++, змінних, констант, операторів, виразів.

2 Завдання та порядок виконання

2.1 Вивчити теоретичний матеріал.

2.2 Підготувати відповіді на контрольні запитання.

2.3 Записати мовою С++ математичні вирази.

3 Контрольні запитання

3.1 Визначити поняття «ідентифікатор». Правила складання ідентифікаторів.

3.2 Змінні. Типи, діапазони їх значень, правила оголошення змінних.

3.3 Константи. Вигляди констант.

3.4 Основні класи операторів.

3.5 Визначити поняття «блок».

3.6 Визначити поняття «препроцесор мови С++». Директиви *#include, #define*.

4 Зміст звіту

4.1 Номер роботи, її назва, визначення мети.

4.2 Стислий зміст теоретичного матеріалу та відповіді на контрольні запитання.

4.3 Результати виконання завдання.

4.4 Висновки до роботи.

5 Навчальний матеріал

5.1 Ідентифікатори. Ідентифікатори – імена, що використовуються для звернення до змінних, функцій, міток та інших визначених користувачем об'єктів. Ідентифікатор може містити від одного до декількох символів. Перший символ обов'язково повинен бути літерою латинського алфавіту чи літерою підкреслення. Значущими символами ідентифікаторів С++ є тільки перші 32. Це означає, що якщо ідентифікатори двох змінних мають однакові перші 32 символи та розрізняються тільки з 33-го символу, то С++ ці змінні не розрізняє.

У мові С++ прописні та рядкові літери трактуються як різні. Наприклад, ідентифікатори *count*, *Count* і *COUNT* представляють три різні змінні. Ідентифікатор змінної не повинен збігатися з ключовим словом, з ім'ям бібліотечної чи функції користувача.

5.2 Змінні. Всі змінні перед використанням повинні бути явно оголошені.

У мові С++ використовуються дані п’ятьох типів: символи, цілі числа, числа з плаваючою крапкою, числа з плаваючою крапкою подвійної точності та змінні без значення. Ключовими словами для оголошення змінних цих типів є *char, int, float, double* та *void* відповідно. Нижче наведені діапазони значень для змінних цих типів.

Тип змінної	**Діапазон значень**
char	від -128 до 127
int	від -32768 до 32767
float	від 3.4e-38 до 3.4e+38
double	від 1.7e-308 до 1.7e+308
void	значень не має

Оголошення змінних. Формат оператора оголошення змінних має вигляд

<тип> <список_змінних>;

де *тип* - будь-який допустимий тип змінних;

список_змінних - один чи більше ідентифікаторів, розділених комами.

Наприклад: *int i, j, l;*
double less, roz;

Тут змінні *i, j, l* оголошуються як змінні цілого типу, а змінні *less, roz* - як числа з плаваючою крапкою подвійної точності.

В залежності від місця оголошення змінних вони мають різні галузі використання. Змінні, що оголошені поза всіма функціями, включаючи функцію *main()*, називаються

глобальними і можуть використовуватися всіма блоками програми. Змінні, що оголошені всередині функцій, називаються локальними і можуть використовуватися тільки всередині даної функції. Змінні також можуть оголошуватися у місці оголошення формальних параметрів функції. (Формальний параметр визначає змінну в функції, що набуває значення аргументу, що міститься у виклику функції). Формальні параметри працюють як звичайні локальні змінні.

Рядкові константи. Крім зазначених видів констант існує ще один вид - рядковий. **Рядок** - це набір символів, взятих у подвійні апострофи, наприклад *"це текст"*.

5.3 Константи. Константами вважаються величини, що не змінюються у процесі виконання програми. Основні види констант:

Тип даного	**Вид константи**
char	*'a', '\n', '9'*
int	*1, 123, 21000*
float	*12.23, 4.34e-3*
double	*12345212, 1.0e100*

Не плутайте рядкові константи із символьними: перші беруться у подвійні апострофи, другі - в одинарні.

Символьні константи із зворотним слешем. Є символи, що не можна ввести з клавіатури, наприклад, повернення каретки. Для таких символів передбачені спеціальні символьні константи із зворотним слешем:

f - встановлення на нову сторінку;

n - встановлення на новий рядок;

r - повернення каретки;

t - горизонтальна табуляція;

v - вертикальна табуляція.

Ініціалізація змінних. Змінним можна присвоїти значення під час їх оголошення. Загальний формат ініціалізації такий:

<тип> <ім'я_змінної> = <константа>.

Наприклад: *char ch = 'a';*
int first = 0;

5.4 Оператори. У мові С++ існують три основних класи операторів:

- арифметичні;
- логічні;
- оператори порівняння.

Арифметичні оператори:

Оператор	Дія
+	Додавання
–	Віднімання та знак «мінус»
*	Множення
/	Ділення
%	Цілочисельне ділення (дає остачу від ділення цілих чисел)
++	Додатний приріст
––	Від'ємний приріст

Додатний та від'ємний приріст. Оператор «++» додає до свого операнда 1, а оператор «––» віднімає 1. Таким чином, оператори *x++;* та *x––;* еквівалентні операторам *x=x+1;* та *x=x–1;*, тільки значно компактніше.

Оператори «++» та «––» можуть стояти перед або після операндів, наприклад, оператор *x=x+1;* можна записати як *++x;* або як *x++;*.

Однак зазначені оператори працюють по-різному. Якщо оператори додатного чи від'ємного приросту знаходяться попереду операнда, то спочатку виконується операція приросту і після цього використовується значення операнда. Якщо оператор приросту знаходиться після операнда, то спочатку використовується поточне значення операнда і після цього виконується приріст. Наприклад, у випадку

x=10; *y=++x;*

С++ присвоює *y* значення *11*, так як спочатку виконується приріст *x* і після цього його значення присвоюється *y*. Однак, у випадку

x=10; *y=x++;*

y присвоюється значення *10*, а після цього виконується збільшення значення *x*.

Пріоритет арифметичних операцій такий:

найбільший ++, ––

–

*, /, %

найнижчий +, –

Послідовність операторів одного пріоритету виконується зліва направо. Для зміни цього порядку можна використовувати дужки. Вираз у дужках виконується в першу чергу.

Оператори порівняння та логічні оператори. Оператори порівняння дозволяють визначити співвідношення двох величин. Логічні оператори встановлюють взаємозв'язок цих співвідношень.

В основі роботи логічних операторів та операторів порівняння лежать поняття «істинно» та «хибно». Істінним вважається будь-який вираз, відмінний від нуля, нульове значення – «хибно».

Оператори порівняння	
Оператор	Дія
>	більше ніж
>=	більше або дорівнює
<	менше ніж
<=	менше або дорівнює
==	Дорівнює
!=	не дорівнює

Логічні оператори	
Оператор	Дія
&&	AND
||	OR
!	NOT

Всі оператори порівняння та логічні оператори мають пріоритет нижче, ніж арифметичні оператори.

Пріоритети логічних операторів та операторів порівняння:

найбільший !

>, >=, <, <=

==, !=

&&

найнижчий ||

Як і в разі арифметичних виразів, порядок виконання цих операторів можна змінювати за допомогою дужок.

5.5 Вирази. Виразом у мові С++ вважається будь-яке допустиме поєднання операторів, констант та змінних.

Оскільки більшість виразів збігається з алгебраїчними формулами, правила запису та виконання виразів звичайно приймають такими ж. Однак, існують і свої особливості.

Перетворення типів у виразах. Якщо в арифметичному виразі зустрічаються операнди різних типів, то один з операндів підлягає перетворенню типів так, щоб він відповідав типу іншого операнда. Операнд для перетворення вибирається за наступним правилом. В С++ основні типи мають визначений порядок старшинства, що визначає, який операнд перетворюється, а який ні. Якщо дивитися справа наліво, то порядок старшинства типів такий:

символьні < цілі < плаваючі < подвійної точності.

Типи, які знаходяться праворуч, перевищують за старшинством всі типи, що знаходяться ліворуч.

5.6 Основні математичні функції. Основні математичні функції наведені в таблиці 2.1.

Таблиця 2.1 – Основні математичні функції

Позначення функції в мові С++	Пояснення
1	2
sin(x)	Функція синусу. Обчислює синус аргументу
cos(x)	Функція косинусу. Обчислює косинус аргументу
tan(x)	Функція тангенсу. Обчислює тангенсу аргументу
log(x) log10(x)	Функція натурального логарифму. Обчислює натуральний логарифм додатного аргументу Логарифм по основі 10
abs(x)	Функція - абсолютна величина. Обчислює абсолютне значення (модуль) цілого аргументу
fabs(x)	Функція - абсолютна величина. Обчислює абсолютне значення (модуль) аргументу з плаваючою крапкою
exp(x)	Експоненціальна функція. Обчислює *e* в ступені *x*, де e ≈2.71828182845904523536

Продовження таблиці 2.1

1	2
sqrt(x)	Функція квадратного кореня. Обчислює квадратний корінь додатнього аргументу
pow(x,y)	Функція ступеня. Обчислює *x* в ступені *y*
acos(x) asin(x) atan (x) atan 2(x)	Обчислює арккосинус аргументу arcos x Обчислює арксинус аргументу arsin x Обчислює арктангенс аргументу arctg x

Для виклику функцій необхідно підключити бібліотеку *<math.h>* за допомогою директиви *#include <math.h>*. Про цю директиву буде сказано нижче.

Приклад запису функцій :

Математичний запис	Запис мовою С++
$z = sin(x)^2$	*z = pow(sin(x),2);*
$z = sin(x^2)$	*z = sin(pow(x,2));*

5.7 Структура програми мовою С++. Мова С++ побудована на концепції складених блоків, що називаються функціями. Програма складається з однієї чи більше функцій. При написанні програми пропонується спочатку створити функції, а після цього об'єднати їх.

Кожна функція являє собою програму, що містить один чи більше операторів і виконує одну чи декілька задач. У добре складеній програмі кожна функція виконує тільки одну задачу.

У загальному випадку функція має такий формат:

```
<тип_значення,_що_повертається> <ім'я_функції> (<параметри>)
{
  <оголошення змінних>;
  <тіло програми>;
}
```

Всі програми мовою С++ повинні мати функцію *main()*, так як саме з неї починається виконання програми. Програма може містити одну і тільки одну функцію з ім'ям *main()*.

5.8 Характерні особливості мови С++. Крапка з комою - обмежувач оператора. Таким чином, кожний окремий оператор закінчується цим символом.

С++ не розпізнає кінця рядка як обмежувач. Це означає, що для наочності можна групувати оператори на одному рядку, як показано нижче.

```
x = y;
y = y+1;
mul(x, y);
```

це те саме, що і

```
x = y;  y = y+1;  mul(x, y);
```

Для зручності читання програми можна розміщати прогалини у будь-яке її місце.

Блок - сукупність логічно зв'язаних операторів, взятих у дужки. Введення додаткових дужок не викличе помилок і не сповільнить обчислення виразів. Додаткові дужки використовуються для того, щоб був ясний порядок обчислення виразів.

Розглянемо програму:

```
// Програма номер 1

#include <stdio.h>
main()
{
int  aud;
aud = 222;
printf("Номер аудиторії %d\n", aud);
}
```

Перший рядок. В С++ коментарі починаються із символів //. Все, що знаходиться після зазначених символів, до кінця поточного рядку компілятор С++ ігнорує. Для виділення блока коментарів використовують символи /* на початку блока та символи */ в кінці блока коментарів.

Другий рядок - пустий рядок. Пусті рядки дозволені, вони не виявляють ніякого впливу на виконання програми.

Третій рядок - *#include <stdio.h>* - директива препроцесора, що підключає стандартну бібліотеку ввдення/виведення мови С++. Директива *#include* буде розглянута нижче.

П'ятий рядок *main()* визначає ім'я функції. Всі програми в C++ починають виконуватися з виклику основної функції *main()*. Виконання програми припиняється по досягненні кінця функції *main()*.

Наступний рядок складається з однієї фігурної дужки {, що позначає початок основної функції *main()*.

Першим рядком програми всередині функції *main()* є *int aud;* - це є оголошення змінної з ім'ям *aud*, що може набувати тільки цілих значень. Наступний рядок

aud = 222;

є оператором присвоювання. Цей оператор присвоює значення *222* змінній *aud*.

Наступний рядок виводить інформацію на екран:

printf ("Номер аудиторії %d\n ", aud);

В C++ немає операторів введення/виведення. Замість цього існують функції введення/виведення, прототипи яких знаходяться у стандартній бібліотеці <*stdio.h*>. Бібліотека підключається за допомогою директиви препроцесора *#include* і викликається по мірі необхідності. Виклик функції виконується просто: достатньо вказати ім'я функції і записати необхідні атрибути (аргументи).

Виклик функції *printf()* буде розглянутий при вивченні операторів введення/виведення.

В останньому рядку програми знаходиться фігурна дужка, яка закривається, що говорить про завершення основної функції *main()*.

Так, як створюється функція *main()*, можна створювати і інші функції та викликати їх з інших частин програми. Наприклад, у поданій нижче програмі використовується функція *hello()* для друку на екрані слова *hello()*:

```
// Програма, що використовує дві функції

#include <stdio.h>
main()
{
```

```
void hello(void);// прототип функції hello()
hello();              // виклик функції hello()
}

void hello(void)  // функція hello()
{
printf("hello");
}
```

Звичайно функція описується до того, як вона буде визначена. Опис інформує компілятор про існування функції, про тип значення, що повертається, а також про тип параметрів, що їй передаються. Опис функції часто називають прототипом функції. Опис функції має такий вигляд:

<тип_значення,_що_повертається> <ім'я_функції> (<параметри>);

5.9 Препроцесор мови С++. У програмах мовою С++ широко використовуються можливості препроцесора. **Препроцесор** – це програма, що обробляє вхідний модуль до того, як він пройде через компілятор, заміняючи визначені імена еквівалентними їм рядками.

Рядки, що містять директиви препроцесора, розпочинаються з символу номера *(#)*.

Директива *#include*. Директива *#include* здійснює підстановку замість себе тексту, зазначеного у директиві файлу. Вхідний файл, що буде читатися, повинен бути взяти у подвійні лапки або гострі дужки. Наприклад:

```
#include "stdio.h"
#include <stdio.h>
```

Якщо ім'я файлу взято у лапки, то компілятор спочатку буде шукати ім'я файлу у поточному робочому каталозі . Якщо компілятор не знаходить файл, то він шукає ім'я файлу у будь-якому каталозі, який специфіковано у командному рядку.

Укладення у гострі дужки ім'я файлу каже про те, що пошук цього файлу буде здійснюватися у специфікованому каталозі C++. Якщо компілятор не знаходить файл, то пошук буде виконуватися у стандартному каталозі.

У мові C++ файли з поширенням *.h* називаються файлами-заголовками (Header File). Вони містять опис змінних, функцій, типів, що використовується багатьма програмами. В даному випадку у файлі *stdio.h* міститься опис, необхідний для використання стандартної бібліотеки введення/виведення мови C++. Ім'я файлу дісталося від скорочування *Standard Input/Output*. Цей файл буде включатися перед всіма програмами, де є введення або виведення.

Директива *#define*. Директиву препроцесора *#define* застосовують для того, щоб визначити ідентифікатор та ланцюжок, який компілятор буде підставляти замість ідентифікатора кожний раз, коли він зустрічається у вхідному файлі. Ідентифікатор називається макроіменем, а процес підстановки називається макропідстановкою. Загальний формат цієї директиви має вигляд

#define <ідентифікатор> <ланцюжок>

Наприклад, для використання *True* у вигляді значення *1* та *False* як значення *0*, необхідно оголосити два *#define* так, як показано нижче:

```
#define True 1
#define False 0
```

Ці рядки змушують компілятор підставляти *1* або *0* кожний раз, коли він зустрічає у вхідному файлі *True* або *False*.

Наприклад, дана функція *printf()* буде друкувати на екрані *0 1 2*:

```
printf(" %d %d %d", False, True, True+1);
```

Після того, як макроім'я визначено, його можна використовувати у вигляді частини визначення інших макроімен. Наприклад:

```
#define One 1
#define Two One+One
#define Three One+Two
```

Лабораторна робота 3. Програмування лінійних обчислювальних процесів

1 Мета роботи: вивчення оператора присвоювання, функцій введення/виведення. Набуття навичок складання програм лінійних обчислювальних процесів та виконання їх на ПЕОМ.

2 Завдання та порядок виконання

2.1 Вивчити теоретичний матеріал.

2.2 Підготувати відповіді на контрольні запитання.

2.3 Скласти програми мовою С++, згідно зі складеними алгоритмами по лаб. роботі 1 методичних вказівок 981[5] або 54[6].

2.4 Ввести підготовлені програми у ПЕОМ, виконати їх та отримати результати розрахунків.

3 Контрольні запитання

3.1 Структура програм мовою С++.

3.2 Формат запису оператора присвоювання.

3.3 Формат запису функції введення. Коди формату.

3.4 Формат запису функції виведення. Коди формату, модифікатори.

4 Зміст звіту

4.1 Номер роботи, її назва, визначення мети.

4.2 Стислий зміст теоретичного матеріалу та відповіді на контрольні запитання.

4.3 Результати виконання завдання: схеми алгоритмів, програми, результати роботи програм.

4.4 Висновки до роботи.

5 Навчальний матеріал

5.1 Оператор присвоювання. Оператор присвоювання зображається знаком рівності «=». Він присвоює значення, що отримано у правій частині, змінній у лівій частині виразу.

Формат оператора присвоювання:

<ім'я_змінної> = <значення>;

Наприклад: aud = 222;

5.2. Функція *scanf()*. Універсальною функцією введення є функція *scanf()*. Вона може читати всі типи даних і автоматично перетворює числа у належний внутрішній формат.

Формат функції *scanf()*:

scanf ("<управляючий_рядок>", <список_аргументів>);

Список аргументів повинен містити стільки ж аргументів, скільки специфікацій формату знаходиться в управляючому рядку.

В управляючому рядку перед специфікаторами формату введення, які повідомляють функції *scanf()* тип даних, що будуть читатися, розташовується знак %.

Коди формату функції *scanf()*:

Код	Значення
%d	Читати десяткове ціле
%f	Читати число з плаваючою крапкою
%c	Читати окремий символ
%s	Читати рядок символів
%o	Читати вісімкове число
%x	Читати шістнадцяткове число
%p	Читати покажчик

Іменам змінних, що отримують значення шляхом введення з клавіатури, повинен передувати знак *&*. (Всі змінні, які використовуються для набуттязначень шляхом функції *scanf()*, повинні передаватися їх адресами. Це означає, що всі аргументи повинні бути покажчиками на змінні, які використовуються у вигляді аргументів).

Наприклад, для того, щоб зчитати ціле у змінну *count*, можна використати такий виклик функції *scanf()*:

scanf("%d", &count);

читання двох значень у дві змінні відповідно

scanf("%d,%d", &i, &j);

5.3 Функція *printf()*. Для форматного виведення використовується функція *printf()*. Її формат

printf("<управляючий_рядок>", <список_аргументів>).

Тут «управляючий_рядок» містить або символи, що виводяться на екран, або специфікації формату, які визначають засіб подання аргументів, що виводяться, або те й інше. Нижче наводяться допустимі специфікації формату.

Код	Значення
%d	Виводити десяткове ціле
%f	Виводити число з плаваючою крапкою
%c	Виводити окремий символ
%s	Виводити рядок символів
%o	Виводити вісімкове число
%x	Виводити шістнадцяткове число
%p	Виводити покажчик

Специфікації формату можна включати у будь-яке місце управляючого рядка. При викликанні функції *printf()* проглядає управляючий рядок і запам'ятовує специфікації формату. Після цього вона виводить на екран символи у порядку їх появи, а при зустрічі аргументу виводить його відповідно до специфікації формату. При цьому функція ставить у взаємно однозначну відповідність зліва направо специфікації формату та аргументи. Число специфікацій формату визначає число очікуваних аргументів. Наприклад:

printf (" %s %d", "це рядок", 100);
виводить на екран: *це рядок 100*

printf ("це рядок %d", 100);
виводить на екран: *це рядок 100*

printf ("число %d - десяткове, %f - дробове.", 10, 110.789);
виводить на екран: *число 10 - десяткове, 110.789 - дробове.*

Команди формату можуть мати модифікатори, які специфікують ширину поля, кількість десяткових розрядів та флажок вирівнювання по першому лівому знаку чи розряду. Ціле, що розташовується між знаком % і командою формату, діє як специфікатор мінімальної ширини поля. Цей специфікатор змушує ПЕОМ доповнювати виведення пропусками чи нулями, для того, щоб забезпечити визначену мінімальну його довжину. Якщо ланцюжок символів чи число перевищує цей мінімум, то функція *printf()* буде друкувати їх повністю, навіть якщо вони виходять за межу цього мінімуму. По умовчанню, для доповнення виведення використовуються пропуски. Щоб доповнити виведення нулями, необхідно розташувати *0* перед специфікатором ширини поля. Наприклад, *%05d* буде доповнювати число, яке складається менш ніж з п’ятьох цифр, нулями перед значенням числа, для того щоб загальна довжина була рівною п'ятьом.

Щоб специфікувати кількість десяткових розрядів, які необхідно надрукувати в разі числа з десятковою крапкою, десяткова крапка розміщується після специфікатора ширини поля та за нею - кількість десяткових розрядів. Наприклад, *%10.4f* буде виводити число, загальне поле виведення якого *10* позицій, з них чотири позиції - дробова частина та одна позиція - десяткова крапка. Коли формат, подібний цьому, застосовується до ланцюжків символів чи цілих, число, що слідує за крапкою, специфікує максимальну довжину поля. Наприклад, *%5.7s* буде виводити ланцюжок символів довжиною не менше п'ятьох символів та не більше ніж сім символів. Якщо ланцюжок більший, ніж максимальна ширина поля, то символи, що залишилися будуть відсічені.

За умовчанням, усе виведення вирівнюється праворуч: якщо ширина поля більше ніж дані, що друкуються, то такі дані розташовуються на правому краю цього поля. Можна викликати вирівнювання інформації ліворуч, розташовуючи знак «мінус» безпосередньо після *%*. Наприклад, *%-10.2f* буде вирівнювати ліворуч число з плаваючою крапкою з двома десятковими розрядами у десятипозиційному полі.

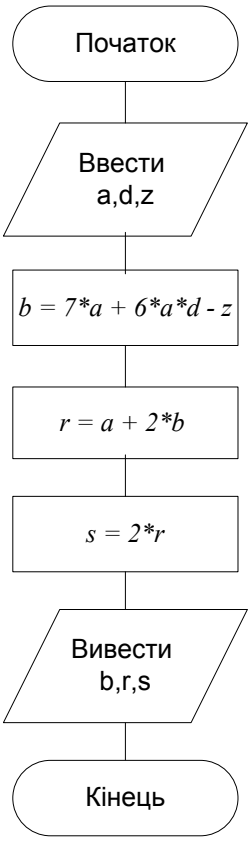
Деякі приклади управляючих рядків для виведення даних:

функція printf()	виведення
("%9.2f", 123.321)	123.32
("%-9.2f", 123.321)	123.32
("%5.7s", "123456789")	1234567
("%9d", 555)	555
("%09d", 555)	000000555

Приклад

Скласти схему алгоритму та програму для обчислення функції:

*s = 2*r, якщо r = a + 2*b + ab, b = 7*a + 6*a*d – z*




```
#include <stdio.h>
#include <math.h>
main()
{float a,b,d,r,s,z;
printf("Запровадьте значення a,d,z:");
scanf("%f %f %f",&a,&d,&z);

b=7*a+6*a*d-z;

r=a+2*b+sqrt(a*b);

s=2*pow(r,fabs(a));

printf("Значення s= %f,b= %f,r= %f",s,b,r);
}
```

Лабораторна робота 4. Організація управління обчислювальним процесом

1 Мета роботи: вивчення операторів передачі управління. Набуття навичок складання програм розгалужених обчислювальних процесів та виконання їх на ПЕОМ.

2 Завдання та порядок виконання

2.1 Вивчити теоретичний матеріал.

2.2 Підготувати відповіді на контрольні запитання.

2.3 Скласти програми мовою С++, згідно зі складеними алгоритмами по лаб. роботі 2 методичних вказівок 981[5] або 54[6].

2.4 Ввести підготовлені програми у ПЕОМ, виконати їх та отримати результати розрахунків.

3 Контрольні запитання

3.1. Формат та логіка дії оператора *if*.

3.2. Який вираз мовою С++ є істиним?

3.3. Чи можуть в операторі *goto* використовуватися такі мітки: *10, m1, мітка10, _20m, 20met*?

4 Зміст звіту

4.1. Номер роботи, її назва, визначення мети.

4.2 Стислий зміст теоретичного матеріалу та відповіді на контрольні запитання.

4.3 Результати виконання завдання: схеми алгоритмів, програми, результати роботи програм.

4.4 Висновки до роботи.

5 Навчальний матеріал

5.1 Оператор умовної передачі управління *if*. Формат оператора if має вигляд

if (<умова>) <оператор>;
[else <оператор>];

Якщо умова виконується («істинна» або будь-який знак відмінний від нуля), то комп'ютер виконає оператор чи блок операторів, наступний за оператором if. У противному випадку, якщо існує частина *else*, виконується оператор чи блок операторів, наступний за *else*.

У вигляді умовного виразу оператора *if* можна використовувати будь-який дозволений у С++ вираз, а не тільки вирази порівняння та логічні вирази. Вираз в операторі *if* аналізується та розрізняється: нуль або ненульове значення.

Приклад. Програма виводить на екран повідомлення «Правильно», якщо вгадано задумане число, та «Неправильно», якщо число не вгадано.

```
#include <stdio.h>
main()
{
  int  magic;
  int  guess;
  magic=5;    // задумане число
  printf (" Ваше число: ");
  scanf ( "%d", &guess );
  if (guess==magic)  printf ("  ** Правильно **\n ");
  else  printf ("  **Неправильно **\n ");
}
```

Якщо при виконанні умови необхідно виконати групу операторів, то використовують блок:

```
if (x) {

група операторів
}
else {

група операторів
}
```

Укладені оператори *if*. Укладений оператор *if* може слідувати або за частиною *if*, або за частиною *else*. Складність в тому, що важко одразу сказати, якому *if* відповідає який *else*.

Розглянемо, наприклад, такий фрагмент:

```
if (x)
if (y)  printf (" 1 ");
else  printf (" 2 ");
```

На який *if* посилається *else*? Є просте правило: *else* пов'язане з ближчим *if*, що не має парного *else*. І *if*, і *else* повинні знаходитися всередині одного блока. В даному випадку *else* пов'язане з оператором *if(y)*. Для того, щоб пов'язати *else* з оператором *if(x)*, необхідно використати фігурні дужки:

```
if (x) {
if (y)  printf (" 1 ");
}  else printf (" 2 ");
```

5.2 Оператор безумовної передачі управління *goto*. Формат оператора *goto* має вигляд

goto <метка>;

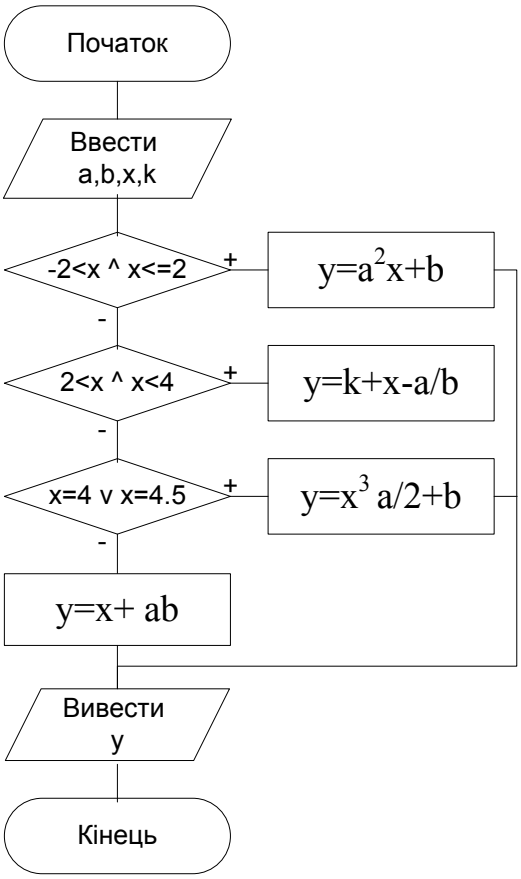
Для роботи оператора *goto* необхідна наявність мітки. Мітка - це повноправний ідентифікатор, за яким слідує двокрапка. Мітка повинна знаходитися у тій же функції, що і оператор *goto*.

Наприклад, можна організувати цикл, що працює 100 раз, за допомогою операторів *if* та *goto*:

```
        x=1;
loop:
        x++;
        if(x<100)  goto loop;
```

Приклад 1. Скласти схему алгоритму та програму обчислення значення функції y:

$$y = \begin{cases} a^2 x + b, & \text{при } -2 < x <= 2 \\ k + x - a/b, & \text{при } 2 < x < 4 \\ x^3 a/2 + b, & \text{при } x = 4 \text{ або } x = 4.5 \\ x + ab, & \text{в інших випадках.} \end{cases}$$




```
#include <stdio.h>
#include <math.h>
main()
{
float a,b,x,k,y;
printf("Ввести a,b,x,k:");
scanf("%f%f%f%f",&a,&b,&x,&k);
if(x>-2 && x<=2)
 y=pow(a,2)*x+b;
 else if (x>2 && x<4)
        y=k+x-(a/b);}
       else if (x==4 || x==4.5)
            y=pow(x,3)*a/2+b;
            else y=x+a*b;

printf("y=%f\n",y);
}
```

Лабораторна робота 5. Програмування циклічних обчислювальних процесів

1 Мета роботи: вивчення операторів організації циклічних обчислювальних процесів. Набуття навичок складання програм з циклічними обчислювальними процесами та виконання їх на ПЕОМ.

2 Завдання та порядок виконання

2.1 Вивчити теоретичний матеріал.

2.2 Підготувати відповіді на контрольні запитання.

2.3 Скласти програми мовою С++, згідно зі складеними алгоритмами по лаб. роботі 3 методичних вказівок 981[5] або 54[6].

2.4 Ввести підготовлені програми у ПЕОМ, виконати та отримати результати розрахунків.

3 Контрольні запитання

3.1. Які бувають цикли? У чому їх відмінність?

3.2. Які оператори призначені для організації циклів мовою С++?

3.3. Формат запису циклу *for*.

3.4 Засоби задання приросту у циклі *for*.

3.5 Формат запису циклу *while*.

3.6 Формат запису циклу *do-while*.

3.7 З якою метою використовується оператор переривання циклу?

3.8 З якою метою використовується оператор продовження циклу?

4 Зміст звіту

4.1 Номер роботи, її назва, визначення мети.

4.2 Стислий зміст теоретичного матеріалу та відповіді на контрольні запитання.

4.3 Результати виконання завдання: схеми алгоритмів, програми, результати роботи програм.

4.4 Висновки до роботи.

5 Навчальний матеріал

Цикли дозволяють багаторазово повторювати деякий набір операцій до тих пір, поки не виконається деяка умова. У мові С++ розрізняють цикли *for, while* та *do-while*.

5.1 Цикл *for*. Формат циклу *for*:

for(<ініціалізація>; <умова>; <крок>) <оператор>;

де

ініціалізація - це оператор присвоювання початкового значення управляючій змінній циклу;

умова - вираз порівняння, що перевіряє досягнення управляючою змінною циклу кінцевого значення;

крок - визначає приріст, на який зміниться управляюча змінна при кожному проходженні циклу.

У вигляді прикладу розглянемо програму, яка виводить на екран цифри від 1 до 100:

#include <stdio.h>

```
main ()
{
 int  x;
 for(x=1; x<=100; x++) printf ("%d", x);
}
```

У циклі *for* можна вказати від'ємний прирост для управляючої змінної:

```
#include <stdio.h>
main ()
{
int x;
for (x=100; x>0; x--) printf ("%d", x);
}
```

Використання оператора приросту управляючої змінної не є обов'язковим. Замість нього може стояти будь-який допустимий оператор присвоювання. Наприклад, наступний цикл буде виводити на екран числа від *0* до *100* з кроком *5*:

```
#include <stdio.h>
main ()
{
int  x;
for (x=0; x<=100; x=x+5)  printf ("%d", x);
}
```

Організувавши блок, можна за допомогою циклу *for* багаторазово виконати цілу групу операторів:

```
#include <stdio.h>
main ()
{
int i;
for (i=0; i<100; i++)  {

        група операторів
                        }
}
```

Важливою особливістю циклу *for* є той факт, що на початку циклу завжди перевіряється умова. Якщо умова не виконується, то не виконуються і оператори всередині циклу.

Іншою особливістю є те, що цикл *for* не вимагає визначення всіх частин. У наступному прикладі цикл буде працювати до тих пір, поки не буде введено число 123:

```
for (x=0; x!=123;)  scanf ("%d", &x);
```

При введенні з клавіатури числа 123 умовна частина дає «хибно» і цикл завершується.

Укладені цикли *for*. Укладений цикл *for* може слідувати безпосередньо за умовою зовнішнього циклу:

```
for(i=0; i<10; i++)
    for(j=0; j<8; j++)
       {
        оператори
       },
```

або знаходитися всередині блока

```
for(i=0; i<10; i++)
     {
       оператори
       for(j=0; j<8; j++)
            {
            оператори
            }
     }
```

5.2 Цикл *while*. Формат циклу *while*:

while(<умова>) <оператор>;

де

умова - будь-який допустимий вираз;

оператор - один оператор чи група операторів.

Цикл виконується до тих пір, поки умова істинна. При хибності умови управління передається на оператор, наступний за оператором циклу.

У циклі *while* умова перевіряється на початку циклу. Тобто тіло циклу може взагалі не виконатися.

Приклад. Програма виводить на екран числа від 0 до 4.

```
#include <stdio.h>
main()
{
int i=0;
while(i < 5)  printf("%d\n", i++);
}
```

5.3 Цикл *do-while*. На відміну від циклів *for* та *while*, у циклі *do-while* перевірка умови виконується в кінці циклу. Тому цикл *do-while* завжди виконається хоча б один раз.

Загальний формат цього циклу має вигляд

```
do  {
      <оператор>;
} while(<умова>);
```

Приклад використання циклу *do-while* для читання з клавіатури різноманітних чисел до тих пір, поки одне з них не стане менше за 100:

```
#include <stdio.h>
main()
{
int num;
do {
scanf ("%d", &num);
} while(num>=100);
}
```

5.4 Оператор переривання циклу *break*. Для вимушеного переривання циклу під час його нормального виконання використовують оператор break. При зустрічі оператора break

всредині циклу комп'ютер припиняє виконання циклу і управління передається на оператор, що слідує за даним циклом. Наприклад:

```
#include <stdio.h>
main()
{int  t;
for(t=0; t<100; t++ )
     {
      printf("%d", t);
      if(t==10)  break;       }
}
```

Break здійснює вихід тільки з внутрішнього циклу.

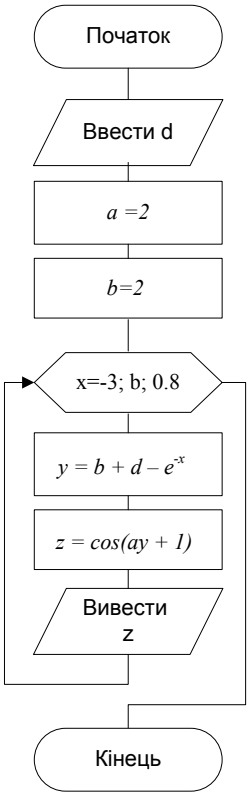
5.5 Оператор *continue*. За дією оператор *continue* подібен оператору *break*, однак він не завершує цикл, а дає команду на наступну ітерацію. При цьому оператори тіла циклу, що залишилися, не виконуються. Наприклад, така програма виводить на екран тільки парні числа:

```
#include <stdio.h>
main ()
{
int  x;
for (x=0; x<100; x++)
      {
        if (x%2)  continue;
        printf ("%d", x);
        }
}
```

Тут, при появі непарного числа, виконується частина if, бо решта від ділення непарного числа на 2 завжди дорівнює 1 або "істині".

Приклад 1. Скласти схему алгоритму та програму обчислення значень функції:

$z = cos(a*y + 1)$, *при* $y = b + d - e^{-x}$, $a = 2$, $b = 2$, $x \in [-3; b]$, $h = 0{,}8$.

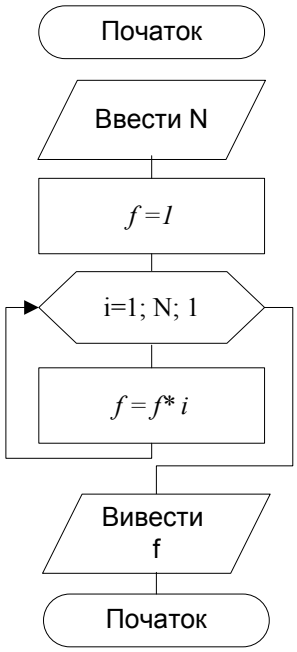



```
#include <stdio.h>
#include <math.h>
#include <conio.h> /* Підключення бібліотеки, необхідної для використання функції clrscr() */
main()
{float a,b,d,x,y,z;

clrscr();  // Функція очищення екрана
printf("Запровадьте d: ");
scanf("%f",&d);
a=2;   b=2;
for (x=-3; x<=b; x=x+0.8)
{  y=b+d-exp(-x);
z=cos(a*y+1);
printf("z=%f \n",z); }
```

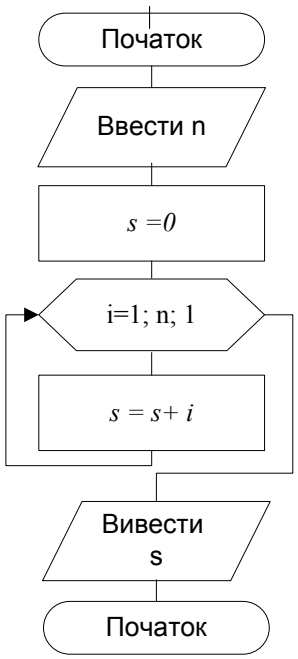
Приклад 2. Обчислити факторіал та суму: f = N! , s = $\sum_{i=1}^{n} i$.

Факторіал




```
#include <stdio.h>
main()
{
 int i,f,N;
 printf("Запровадьте число:");
 scanf("%d",&N);
 f=1;
for (i=1; i<=N; i++)  f=f*i;
 printf("%d ! = %d \n",N,f);
}
```

Сума




```
#include <stdio.h>
  main()
{  int i,s,n;
printf("Запровадьте число:");
   scanf("%d",&n);
 s=0;
for (i=1; i<=n; i++)  s=s+i;
 printf("Сума s = %d \n",s);
}
```

Лабораторна робота 6. Програмування обробки одновимірних масивів

1 Мета роботи: вивчення методики обробки одновимірних масивів. Набуття навичок складання програм для обробки одновимірних масивів та виконання їх на ПЕОМ.

2 Завдання та порядок виконання

2.1 Вивчити теоретичний матеріал.

2.2 Підготувати відповіді на контрольні запитання.

2.3 Скласти програми мовою С++, згідно зі складеними алгоритмами по лаб. роботі 7 методичних вказівок 981[5] лаб. роботі 8 або 54[6].

2.4 Ввести підготовлені програми у ПЕОМ, виконати та отримати результати розрахунків.

3 Контрольні запитання

3.1 Визначити поняття «розмір масиву».

3.2 Визначити поняття «розмірність масиву».

3.3 Формат оголошення одновимірного масиву.

3.4 Функція введення рядка з клавіатури. Особливості її використання.

3.5 Ініціалізація одновимірних масивів.

4 Зміст звіту

4.1 Номер роботи, її назва, визначення мети.

4.2 Стислий зміст теоретичного матеріалу та відповіді на контрольні запитання.

4.3 Результати виконання завдання: схеми алгоритмів, програми, результати роботи програм.

4.4 Висновки до роботи.

5 Навчальний матеріал

5.1 Формат оголошення одновимірного масиву

<тип> <ім'я_масиву> [<розмір>],

де *тип* - визначає базовий тип масиву. Базовий тип визначає тип кожного елементу масиву;

розмір – кількість елементів, що може містити масив.

Наприклад, оголошення масиву цілого типу з ім'ям *sample*, що містить 10 елементів:

```
int  sample [10];
```

Перший елемент масиву отримує індекс *0*. Таким чином, оголошено масив з десятьма елементами: від *sample[0]* до sample[9].

У наступному прикладі організується масив цілого типу, елементам якого присвоюються значення від *0* до *9*:

```
main()
{
int  x[10];
int  t;
for(t=0; t<10; t++)  x[t]=t;
}
```

У мові С++ не здійснюється перевірка меж масиву - ніщо не контролює індекс при виході за межі значення у масиві. Наприклад, якщо перевищення значення індексу масиву трапиться під час

виконання оператора присвоювання, то зайві значення можуть присвоїтися іншим змінним, що розташовані у пам'яті ПЕОМ слідом за полем елементів масиву.

5.2 Рядки. Найбільш часто одновимірні масиви використовуються для створення символьних рядків. Рядок складається з масиву символів, що закінчуються символом *'\0'*. У зв'язку з цим символьний масив повинен містити на один елемент більше ніж містить символів рядок, який записується у нього.

Незважаючи на те, що у мові С++ відсутній тип даних «символьний рядок», мова дозволяє записувавати символьні константи. Згадаємо, що символьна константа - це набір символів, які укладено у подвійні лапки.

5.3 Зчитування рядка з клавіатури. Найкращим засобом введення рядка з клавіатури є використання бібліотечної функції *gets()*. Формат її такий:

gets(<ім'я_масиву>);

Для зчитування рядка необхідно викликати функцію *gets()* з неіндексованим ім'ям масиву як аргумент. Функція *gets()* буде продовжувати зчитувати символи до тих пір, поки не буде натиснута клавіша *Enter*.

Приклад програми, що виводить на екран введений з клавіатури рядок.

```
// Приклад запровадження та друку на екрані рядка

#include <stdio.h>
main()
{
char str[80];
gets (str);
printf ("%s", str);
}
```

У вигляді аргументу функції *printf()* використовується ім'я масиву *str*. Це ім'я не є індексованим. Неіндексоване ім'я масиву можна розміщати там же, де і рядкову константу.

5.4. Виведення рядка на екран. Для виведення рядка на екран використовується бібліотечна функція *puts()*. Формат її такий:

puts(<ім'я_масива>);

Для виведення рядка необхідно викликати функцію *puts()* з неіндексованим ім'ям масиву як аргумент, точно так же, як функцію *gets()*.

Звернення до функції *puts()* вимагає значно менше машинних витрат, ніж таке ж звернення до функції *printf()*, тому що функція *puts()* може тільки виводити ланцюжок символів, вона не може виводити числа чи виконувати перетворення форматів.

5.5 Ініціалізація одновимірних масивів. Загальний формат ініціалізації масиву

<тип> <ім'я_масиву> [<розмір>] = { <список_значень> };

де *список_значень* - це список розділених комами констант, сумісних за типом з базовим типом масиву.

Оператор ініціалізації розмістить першу константу у перший елемент масиву, другу константу - в другий елемент і т. д. У наступному прикладі ініціалізується 10 елементний масив цілих чисел із значеннями від 1 до 10:

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

Тут елемент *i[0]* буде мати значення *1*, елемент *i[9]* - значення *10*.

Символьні масиви, що містять рядки, допускають записувати скорочений формат ініціалізації у вигляді

char <ім'я_масива> [<розмір>] = "<рядок>";

Наприклад, у наступному фрагменті масив *str* ініціалізується рядком *"hello"*:

char str[6] = "hello";

Цей запис еквівалентний запису

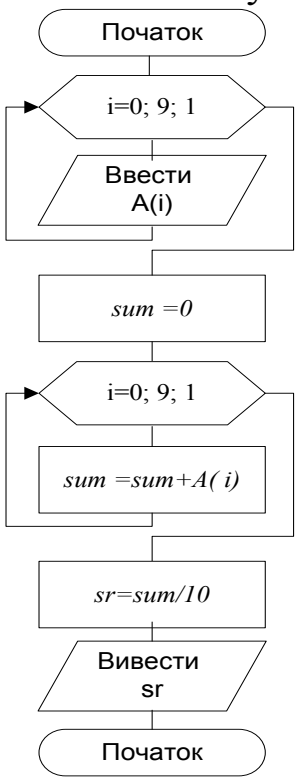
char str[6] = {'h','e','l','l','o','\0'};

Оскільки всі рядки у С++ повинні закінчуватися нулем, масив повинен містити місце і для нього.

5.6 Ініціалізація безрозмірних символьних масивів. Підраховувати вручну кількість символів у кожному повідомленні для завдання розміру масиву дуже трудомістко. Однак можна змусити С++ автоматично оброзмірити масиви за допомогою ініціалізації безрозмірних масивів. Для цього в операторі ініціалізації не слід зазначати розмір масиву, і С++ автоматично створить масив, що зможе містити неявний ініціалізатор. Наприклад:

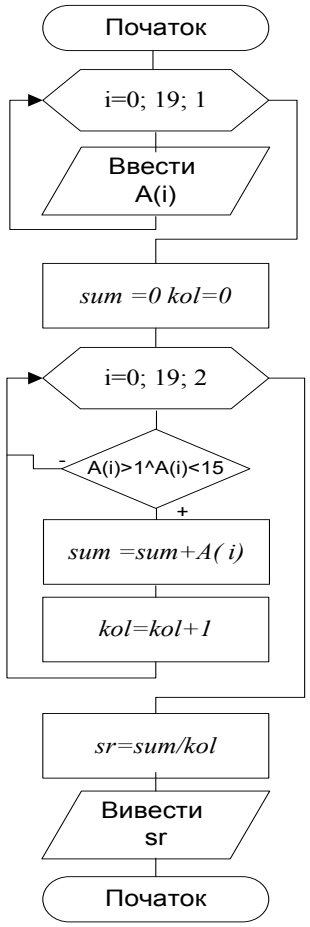
char e[] = "cannot open file\n";

Приклад 1. Обчислити середнє арифметичне значення 10-ти елементів масиву $A=\{A_i\}$, $i=0,...,9$.




```
#include <stdio.h>
#include <conio.h>
main()
{int a[10];
int i,sum;
float sr;
clrscr();
for(i=0; i<10; i++) {
printf("Ввести a[%d]: ",i);
scanf("%d",&a[i]);
}
sum=0;
for (i=0; i<10; i++)
sum=sum+a[i];
sr=(float)sum/10;
printf("Середнє арифметичне
= %f",sr);
}
```

Приклад 2. Обчислити середнє арифметичне значення парних елементів $1 < A_i < 15$ масиву $A=\{A_i\}$, що містить 20 елементів.




```
#include <stdio.h>
#include <conio.h>
main()
{int a[20];
int i,sum,kol;
float sr;
clrscr();
for(i=0; i<20; i++) {
printf("Введіть a[%d]: ",i);
scanf("%d",&a[i]);
}
sum=0;  kol=0;
for (i=0; i<20; i=i+2) {
if(a[i]>1 && a[i]<15) {
sum=sum+a[i];
kol=kol+1; }    }
sr=(float)sum/kol;
printf("Середнє арифметичне =
%f",sr);
}
```

Лабораторна робота 7. Програмування обробки двовимірних масивів

1 Мета роботи: вивчення методики обробки двовимірних масивів. Набуття навичок складання програм обробки двовимірних масивів та виконання їх на ПЕОМ.

2 Завдання та порядок виконання

2.1 Вивчити теоретичний матеріал.

2.2 Підготувати відповіді на контрольні запитання.

2.3 Скласти програми мовою С++, згідно зі складеними алгоритмами по лаб. роботі №8 методичних вказівок 981[5] лаб. роботі 9 або 54[6].

2.4 Ввести підготовлені програми у ПЕОМ, виконати їх та отримати результати розрахунків.

3 Контрольні запитання

3.1 Як визначити кількість елементів двовимірного масиву?

3.2 Які циклічні обчислювальні процеси застосовуються при обробці двовимірних масивів?

3.3 Як у циклі *for* задати обробку елементів масиву, розміщених у колонках з непарними номерами?

3.4 Як виконується ініціалізація двовимірних масивів.

4 Зміст звіту

4.1 Номер роботи, її назва, визначення мети.

4.2 Стислий зміст теоретичного матеріалу та відповіді на контрольні запитання.

4.3 Результати виконання завдання: схеми алгоритмів, програми, результати роботи програм.

4.4 Висновки до роботи.

5 Навчальний матеріал

5.1 Мова С++ дозволяє оголошувати багатовимірні масиви. Найпростішим з багатовимірних масивів є двовимірний масив. Наприклад, для оголошення двовимірного масиву цілих чисел з ім'ям *dvmas* та розмірами *10*20* необхідно записати

```
int  dvmas[10][20];
```

У наступній програмі організується двовимірний масив із значеннями елементів від *1* до *12*.

```
main()
{
int i, j, num[3][4];
for (i=0; i<3; i++)
      for (j=0; j<4; j++)
            num[i][j] = (i*4)+j+1;
}
```

Тут елемент *num[0][0]* буде мати значення *1*, елемент *num[0][1]* - значення *2*, елемент *num[0][2]* - значення *3* і т.д.

5.2 Ініціалізація двовимірних масивів. Двовимірні масиви ініціалізуються так само, як і одновимірні. Наприклад, так можна проініціалізувати двовимірний масив *sqrs* числами від *1* до *10* та їх квадратами:

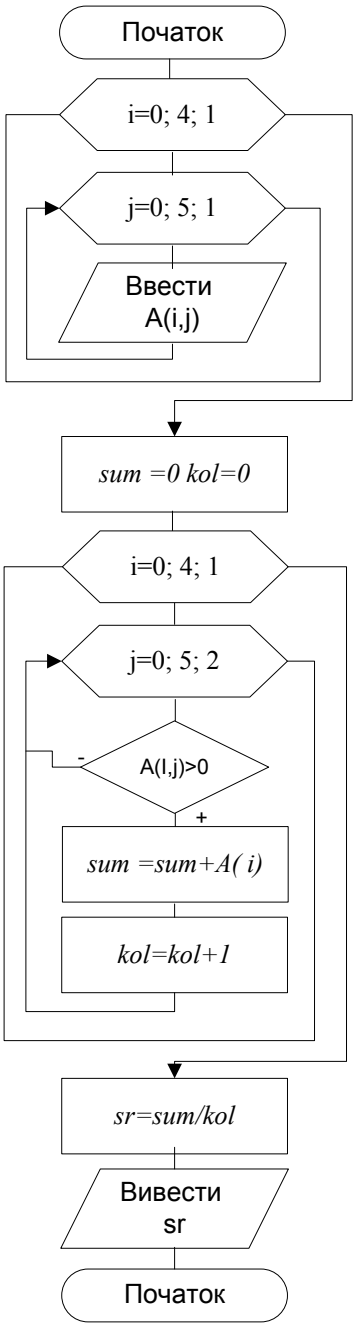
```
int  sqrs[10][2] = {1, 1,
                    2, 4,
                    3, 9,
                    4, 16,
                    5, 25,
                    6, 36,
                    7, 49,
                    8, 64,
                    9, 81,
                    10, 100
                    };
```

5.3 Ініціалізація безрозмірних числових масивів. Ініціалізація безрозмірних масивів не обмежується тільки одновимірними масивами. Однак, для багатовимірних масивів необхідно зазначати всі індекси вимірів, крім найлівішого. Як приклад розглянемо оголошення двовимірного масиву *sqrs* як безрозмірного:

```
int  sqrs[][2] = {1, 1,
                  2, 4,
                  3, 9,
                  4, 16,
                  5, 25,
                  6, 36,
                  7, 49,
                  8, 64,
                  9, 81,
                  10, 100 };
```

Переваги в тому, що можна подовжити або скоротити таблицю не змінюючи індекс по першому виміру.

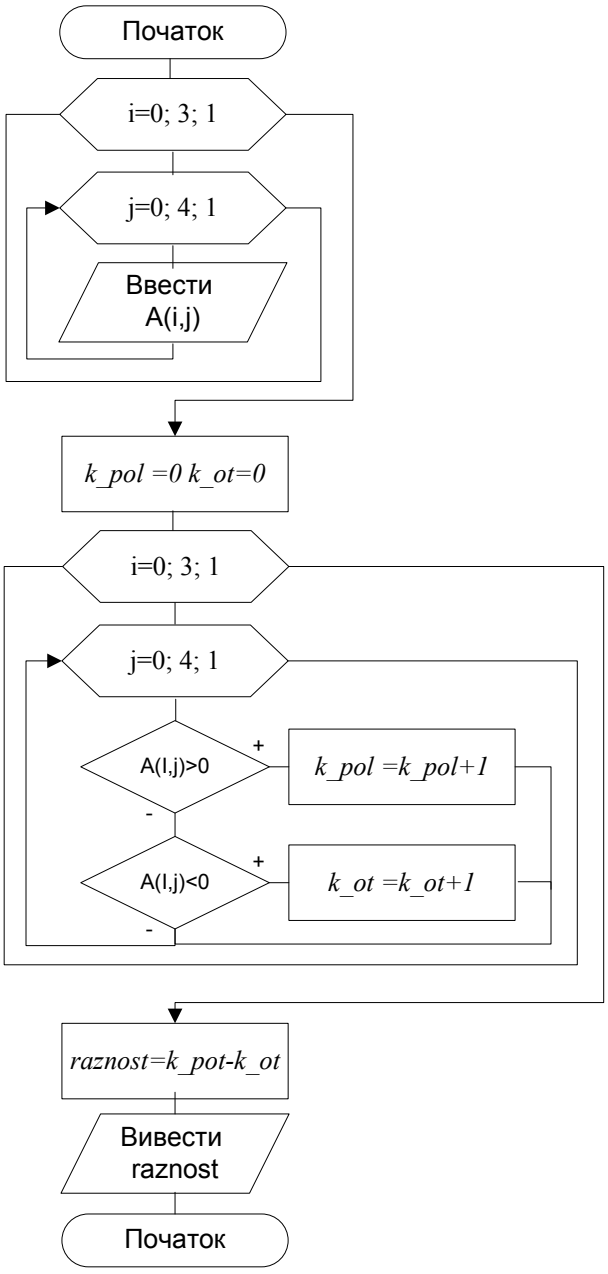
Приклад 1. Задано масив A з елементами a_{ij}, $i=0,...,4$, $j=0,...,5$. Визначити середнє арифметичне значення додатних елементів, розміщених у стовпцях з парними номерами.




```
#include <stdio.h>
#include <conio.h>

main()
{
int a[5][6];
int i,j,summa,kol;
float sr;
clrscr();
for(i=0; i<5; i++)
for(j=0; j<6; j++) {
printf("Ввести a[%d][%d]:",i,j);
scanf("%d",&a[i][j]);
}
summa=0;  kol=0;
for(i=0; i<5; i++)
for(j=0; j<6; j=j+2) {
if(a[i][j]>0) {
summa=summa+a[i][j];
kol=kol+1; }
}
sr=(float)summa/kol;
printf("Середнє арифметичне =
      %f",sr);
}
```

Приклад 2. Задано масив A з елементами a_{ij}, $i=0,...,3$, $j=0,...,4$. Визначити різницю між кількістю додатних та від'ємних елементів.




```
#include <stdio.h>
#include <conio.h>
main()
{
int a[4][5];
int i,j,kol_pol,kol_ot,raznost;
clrscr();
// Введення масиву
for(i=0; i<4; i++)
for(j=0; j<5; j++) {
printf("Ввести a[%d][%d]:",i,j);
scanf("%d",&a[i][j]);
}
// Визначення кількості додатніх
      та від'ємних елементів
kol_pol=0;  kol_ot=0;
for(i=0; i<4; i++)
for(j=0; j<5; j++) {
if(a[i][j]>0)
kol_pol=kol_pol+1;
if(a[i][j]<0)
kol_ot=kol_ot+1;
}
raznost=kol_pol-kol_ot;
printf("raznost=%d",raznost);
}
```

Лабораторна робота 8 Ознайомлення з середовищем програмування

1 Мета роботи: отримання навичок роботи в інтегрованому середовищі програмування Visual C++, ознайомлення з командами основного меню, основними командами редактора.

2 Завдання та порядок виконання

2.1 Вивчити теоретичний матеріал.

2.2 Підготувати відповіді на контрольні запитання.

2.3 Запустити Visual C++ . У вікні редактора набрати приклад навчальної програми. Випробувати команди редактора Visual C++ . Випробувати команди основного меню.

3 Контрольні запитання

3.1 Що містить інтегроване середовище розробки?

3.2 З яких зон складається екран Visual C++ ? Їх призначення.

3.3 Призначення опцій основного меню.

3.4 Команди редактора C++.

3.5 Як створити проект?

4 Зміст звіту

4.1 Номер роботи, її назва, визначення мети.

4.2 Стислий зміст теоретичного матеріалу та відповіді на контрольні запитання.

4.3 Результати виконання завдання: схеми алгоритмів, програми, результати роботи програм.

4.4 Висновки до роботи.

5. Навчальний матеріал

5.1 Запуск C++

Інтегроване середовище розробки (Integrated Development Environment, або скорочено IDE) — це програмний продукт, що поєднує текстовий редактор, компілятор, налагоджувач і довідкову систему.

Будь-яка програма в середовищі Visual C++ завжди створюється у вигляді окремого проекту. Проект (project) — це набір взаємопов'язаних вихідних файлів, компіляція і компоновка яких дозволяє створити програму, що виконується. Основу Visual C++ складає робоча область (project workspace). Вона може містити будь-яку кількість різних проектів, згрупованих разом для узгодженої розробки: від окремого додатка до бібліотеки функцій або цілого програмного пакета. Розв'язання ж простих (навчальних) завдань зводиться до оформлення кожної програми у вигляді одного проекту, тобто робоча область проекту буде містити один проект.

Консольний додаток Після запуску *Visual C++* з'являється головне вікно програми, вигляд якого наведено на рисунку 8.1.

Рис.8.1 – Вигляд робочого вікна *VisualC++*

Екран VisualC ++ розділений на чотири зони. Зверху розташовані меню і панелі інструментів. Крім них робочий стіл Visual C ++ включає в себе три вікна:

a) вікно Project Workspace (вікно робочої області) - розташоване в лівій частині. Спочатку вікно закрите, але після створення нового проекту або завантаження існуючого проекту це вікно буде містити кілька вкладок;

b) вікно Editor (вікно редагування) - розташоване праворуч. Його використовують для введення, перевірки і редагування вихідного коду програми;

c) вікно Output (вікно виведення) служить для виведення повідомлень про хід компіляції, збирання і виконання програми і повідомлень про помилки, що виникають.

Для кнопок панелей інструментів передбачена зручна контекстна допомога: якщо навести курсор миші на кнопку і затримати на пару секунд, то спливе підказка з призначенням даної кнопки. *Developer Studio* дозволяє створювати проекти різних типів, які орієнтовані на різні сфери застосування. Більшість типів проектів є віконними Windows-додатками з відповідним графічним інтерфейсом. Але також передбачена робота і з консольними додатками. Під час запуску консольного додатка операційна система створює консольне вікно, через яке здійснюється введення - виведення даних програми. Така робота і являє імітацію роботи в операційній системі MS DOS або інших операційних системах в режимі командного рядка. Цей тип

додатків найбільше підходить для цілей вивчення мови C / C ++, тому що не потребує створення Windows-коду для інтерфейса користувача.

Створення робочого каталогу

Створіть робочу папку з назвою вашої групи за адресою: C:\Home\ваша_група.

Створення нового проекту

Для створення консольного додатка виконайте такі дії:

- виберіть у рядку меню головного вікна команду *File/New*...
- у діалоговому вікні New виберіть вкладку Projects:
  - оберіть тип проекту: *Win32 Console Application*;
  - введіть ім'я проекту в текстовому полі *Project Name*, наприклад LR_1;
  - в текстовому полі *Location* введіть ім'я каталогу (повний шлях до нього C:\Home\ваша_група) для розміщення майбутніх файлів проекту (якщо вказаний вами каталог відсутній, то він буде створений автоматично). Шлях до майбутнього проекту можна вибрати клацнувши на кнопці, розташованій праворуч від текстового поля Location;
  - клацніть лівою кнопкою миші на кнопці ОК;
- клацання запустить вбудований майстер додатків: Application Wizard, який відкриє діалогове вікно Win32 Console Application - Stepl of 1 з пропозицією визначитися, який підтип консольного додатка бажаєте створити:
  - оберіть тип: *An empty project* (пустий проект);
  - клацніть на кнопці *Finish*;
- після клацання з'явиться вікно: New Project Information (інформація про новий проект) зі специфікаціями проекту та інформацією про каталог, в якому буде розміщений створюваний проект;
- клацніть на кнопці *OK*.

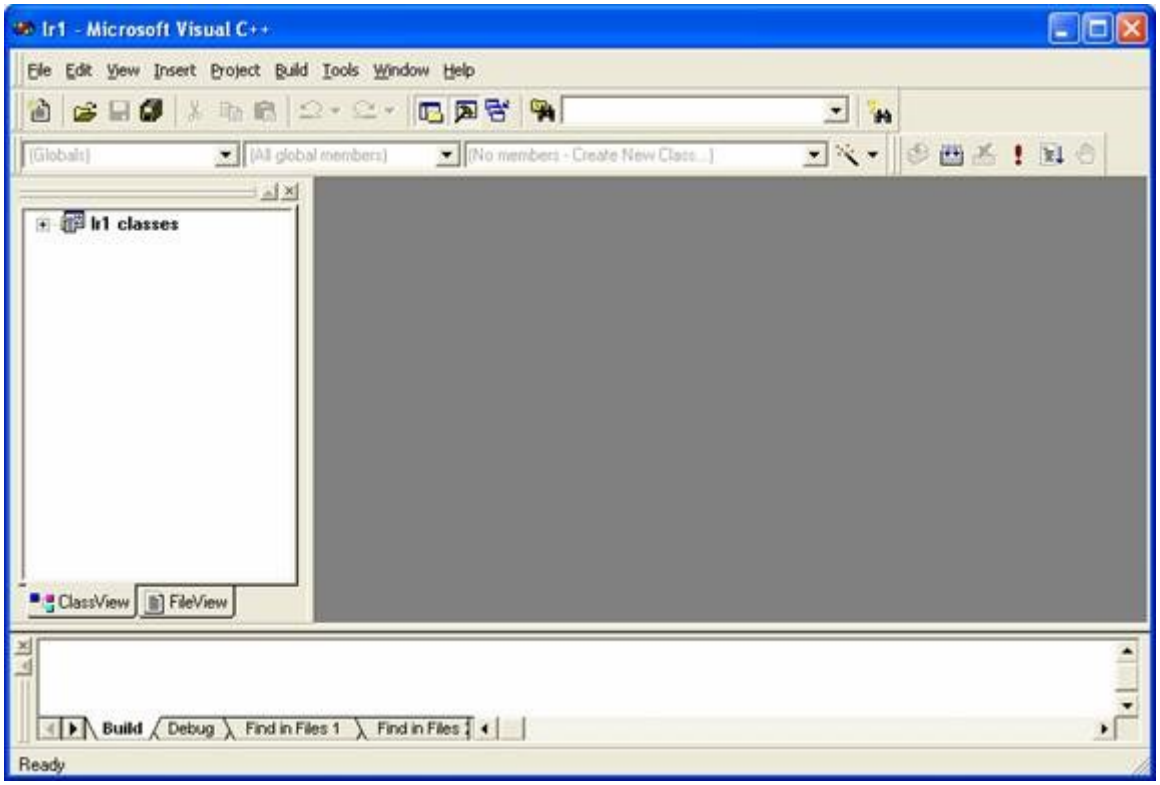


Рис. 8.2 – Робоче вікно після створення консольного додатка

Після виконаних кроків робоче вікно набуде вигляду, який показаний на рисунку 8.2, а в папці LR_1, створеній майстром додатків - файли First.dsw, First.dsp, First.opt, First.ncb і папка Debug (або Release - в залежності від конфігурації проекту).

Короткий опис кожного з файлів:

First.dsw – файл робочої області проекту, який використовується всередині інтегрованого середовища розробки. Він об'єднує всю інформацію про проекти, що входять в дану робочу область;

First.dsp – проектний файл, який використовується для побудови (building) окремого проекту або підпроекту;

First.opt – файл, що містить опції робочої області проекту. Завдяки цьому файлу при кожному відкритті робочої області проекту всі параметри Developer Studio, обрані під час останнього сеансу роботи з даною робочою областю, будуть відновлені.

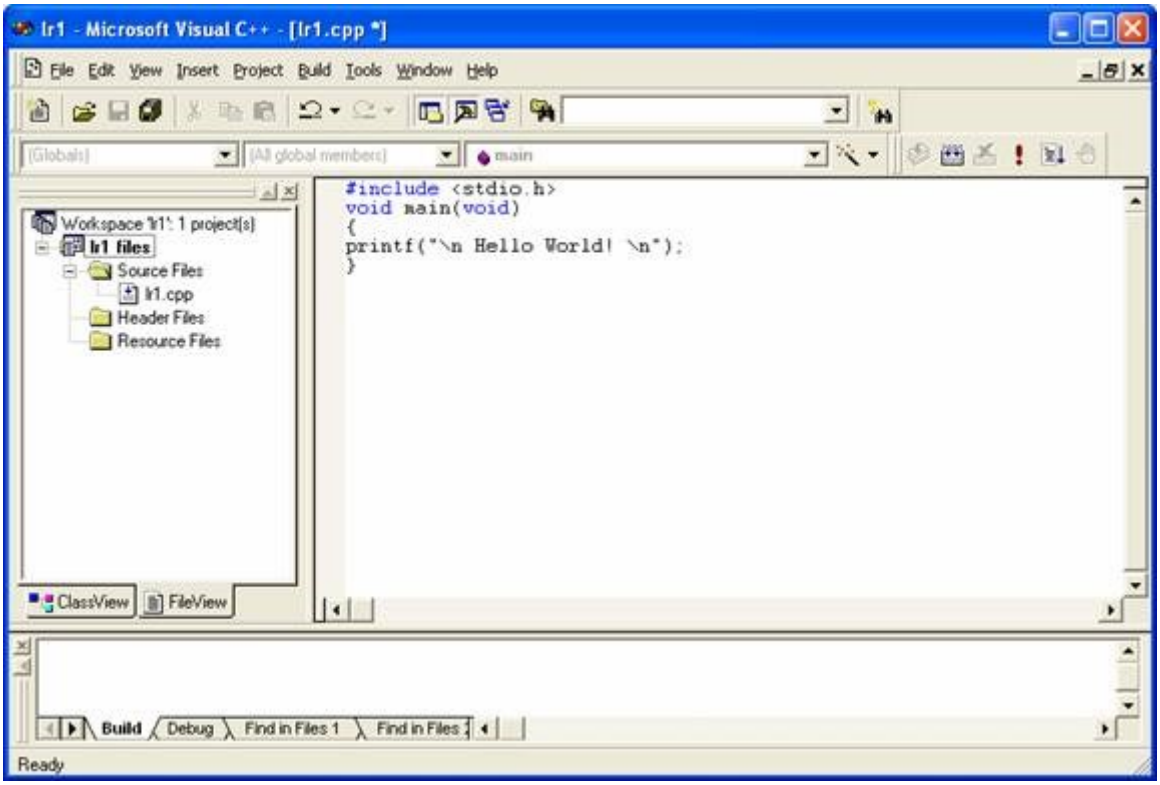


Рис. 8.3 – Вікно додатка після додавання коду

Компіляція, компоновка і виконання проекту. Ці операції можна виконати або через підменю Build головного вікна, або за допомогою кнопок панелі інструментів, або за допомогою комбінації гарячих клавіш. Дане підменю об'єднує команди для компіляції, збирання і налагодження програм.

Основні команди меню Build такі:

1 *Compile* — компіляція обраного файлу. Результати компіляції виводяться у вікно *Output*;

2 *Build* — компоновка проекту. Компілюються всі файли, в яких відбулися зміни з моменту останнього компонування. Після компіляції відбувається складання (link) всіх об'єктних модулів, включаючи бібліотечні, в результуючий виконуваний файл. Повідомлення про помилки компоновки виводяться у вікно *Output*. Якщо обидві фази компонування завершилися без помилок, середовище програмування створить виконуваний файл з розширенням * .exe (для даного прикладу: LR_1.exe), який можна запустити на виконання;

3 *Rebuild All* — те саме, що і *Build*, але компілюються всі файли проекту незалежно від того, чи були них зміни;

4 *Execute* — виконання файлу, створеного в результаті компоновки проекту. Якщо у вихідний текст були внесені зміни – то перекомпіляція, перекомпоновка і виконання.

Операції Compile, Build і Execute - відповідно перша, друга і четверта кнопки панелі інструментів Build MiniBar, яка розташована в робочому вікні (рисунок 8.3), справа вгорі поруч з системними кнопками.

1 Compile = Ctrl+F7;

2 Build = F7;

4 Execute Program = Ctrl+F5.

Скомпілюйте проект. В процесі компіляції у вікні виведення *Output* відображуються діагностичні повідомлення компілятора і збирача. Якщо компіляція відбулася успішно, у вікні виведення отримаємо такий рядок:

LR_1.exe - 0 error(s), 0 warning(s)

Тепер запускаємо додаток на виконання клацанням, наприклад на кнопці Execute Program (Ctrl + F5). З'явиться вікно програми rl1, зображене на рисунку 8.4.

Рисунок 8.4 Результат виконання консольного додатка

Конфігурація проекту. Visual C ++ дозволяє будувати проект або у відлагоджувальній конфігурації (*Win32 Debug*), або в підсумковій конфігурації (*Win32 Release*). Починати роботу потрібно у відлагоджувальній конфігурації, яка зазвичай встановлена за замовчуванням. Для того щоб перевірити, яка поточна конфігурація в проекті, потрібно вибрати в підменю *Project* пункт *Settings*. Відкриється діалогове вікно *Project Settings.* Дивимося, яке значення встановлено у вікні комбінованого списку *Settings For* :. Якщо це не *Win32 Debug*, то перейдіть на потрібне значення через команду меню

Build / Set Active Configuration ... Але у відлагоджувальній конфігурації навіть мінімальний проект має дуже великі розміри. Після отримання фінальної версії проект потрібно переконфігурувати в підсумковій конфігурації (*Win32 Release*).

Як закінчити роботу над проектом? Обрати меню *File*, пункт *Close Workspace*. Або закрити вікно *Visual C++*.

Як відкрити існуючий проект?

- запустити на виконання середовище програмування *Visual C++;*
- обрати в підменю *File*, пункт *OpenWorkspace*;
- у діалоговому вікні знайти папку з потрібним проектом, в ній — файл *ProjectName.dsw* і відкрити знайдений файл клацанням миші.

Якщо проект був у роботі недавно:

- запустити на виконання середовище програмування *Visual C++;*
- обрати підменю *File*, навести курсор миші на пункт *Recent Workspaces*;
- якщо у меню зі списком останніх файлів є файл *ProjectName.dsw*, клацнути по ньому мишею.

Або так:

не викликаючи *Visual C++*, знайти папку з потрібним проектом, в ній — файл *ProjectName.dsw*.

Клацнути мишею на файлі *ProjectName.dsw* і операційна система запустить на виконання середовище *VisualC++*, відкривши при цьому потрібний проект.

Лабораторна робота 9. Програмування лінійних алгоритмів

1 Мета роботи: вивчення правил запису елементарних конструкцій мови програмування С++, змінних, констант, операторів, виразів.

2 Завдання та порядок виконання

2.1 Вивчити теоретичний матеріал.

2.2 Підготувати відповіді на контрольні запитання.

2.3 Записати мовою С++ математичні вирази.

3 Контрольні запитання

3.1 Визначити структуру програми.

3.2 Змінні. Типи, діапазони їх значень, правила оголошення змінних.

3.3 Основні класи операторів.

3.4 Визначити поняття «блок».

3.5 Визначити поняття «препроцесор мови С++». Директиви *#include, #define*.

4 Зміст звіту

4.1 Номер роботи, її назва, визначення мети.

4.2 Стислий зміст теоретичного матеріалу та відповіді на контрольні запитання.

4.3 Результати виконання завдання наданого наприкінці роботи.

4.4 Висновки до роботи.

5 Навчальний матеріал

5.1 Лінійна програма

Якщо в програмі всі оператори виконуються послідовно, один за іншим, така програма називається *лінійною*. Розглянемо як приклад програму, яка обчислює результат за заданою формулою.

Задача 5.1. Розрахунок за формулою

Написати програму, яка переводить температуру в градусах за Фаренгейтом в градуси Цельсія за формулою:

$$C = 5/9\ (F-32),$$

де C – температура за Цельсієм, а F — температура за Фаренгейтом.

Для створення проекту і написання програми використайте матеріали лабораторної роботи 1.

```
#include <iostream.h>
int main()                              //1
{
  float fahr, cels;                                 //2
  cout<<endl<<" Введите температуру По Фаренгейту"<<endl;    //3
  cin>>fahr;                            //4
  cels=5/9 * (fahr - 32);                           //5
  cout<<" По Фаренгейту: "<<fahr<<", в градусах Цельсия: "<<cels<<endl;
  return 0;                                    // 7
}
```

На початку програми записана директива препроцесора, за якою до початкового тексту програми підключаються заголовки <iostream.h>. Це файл, який містить описа операторів введення-виведення cin і cout.

Будь-яка програма на С ++ складається з функцій, одна з яких повинна мати ім'я *main*, яке вказує, що саме з неї починається виконання програми. Після круглих дужок в фігурних дужках {} записується тіло функції, тобто ті оператори, які потрібно виконати. Будь-яка заготовка при написанні програми має вигляд

```
#include <…>
#include <…>
 int main()
{
оголошення змінних;
введення вихідних даних;
розрахунок результату;
виведення результату;
  return 0;
}
```

Для зберігання вихідних даних і результатів треба виділити достатньо місця в оперативній пам'яті. Для цього служить оператор 2. У нашій програмі потрібно зберігати два значення: температуру за Цельсієм і температуру за Фаренгейтом, тому в операторі визначаються дві змінні. Одна - для зберігання температури за Фаренгейтом, названа fahr, інша (за Цельсієм) - cels. Імена змінних дає програміст, виходячи з їх призначення. Ім'я може складатися тільки з латинських букв, цифр і знака підкреслення і повинно починатися з цифри.При описі будь-якої змінної потрібно вказати її тип. Оскільки температура може набувати не тільки цілих значень, для змінних обраний дійсний тип *float*.

Для того щоб користувач (тобто Ви) знав, у який момент потрібно ввести з клавіатури дані, використовується так зване запрошення (оператор 3). На екран виводиться вказаний в операторі *cout* рядок символів, і курсор переводиться в наступний рядок. Для переходу в наступний рядок використовується *endl*.

В операторі 4 виконується введення з клавіатури числа в змінну *fahr*. Для цього виконується стандартний обєкт *cin* і операція зчитування >>. Якщо потрібно ввести декілька величин, використовується ланцюжок операцій >>.

В операторі 5 обчислюється вираз, записаний праворуч від *операції присвоювання* (позначається знаком =), і результат присвоюється змінній cels, тобто заноситься у відведену цій змінній пам'ять. Спочатку ціла константа 5 буде поділена на цілу константу 9, потім результат цієї операції помножений на результат віднімання числа 32 з змінної fahr.

Для виведення результату в операторі 6 застосовується об'єкт *cout.* Виводиться ланцюжок, що складається з п'яти елементів. Це рядок *" За Фаренгейтом:"*, значення змінної *fahr*, рядок *", в градусах Цельсия:"* , значення змінної *cels* і оператор переходу в наступний рядок *endl.*

Останній оператор (7) цієї програми призначений для повернення з неї і передачі значення в зовнішнє середовище.

Скомпілюйте, відлагодьте програму згідно з матеріалами лабораторної роботи 1.

При запуску програми замість російських символів бачимо ???, що викликано різними стандартами кодування символів кирилиці в операційних системах MS DOS-і Windows. Для виправлення додамо в програму функцію CharToOem (доповнення для наочності виділені червоним кольором і курсивом

```
#include <iostream.h>
#include <windows.h>
char  buf[256];
char* RUS(const char* text)
{
  CharToOem(text, buf);
  return buf;
}
 int main()
{   float fahr, cels;
  cout<<endl<<RUS(" Введите температуру По Фаренгейту")<<endl;
  cin>>fahr;
  cels=5/9 * (fahr - 32);
  cout<<RUS(" По Фаренгейту: ")<<fahr;
  cout<<RUS(", в градусах Цельсия: ")<<cels<<endl;
  return 0;
 }
```

Функцію Rus () не можна використовувати більше одного разу в ланцюжку операцій << для одного об'єкта cout, тому ми розбили його на два.

Як ви можете бачити, результат виконання програми зі стабільністю виявляється рівним нулю! Це відбувається через спосіб обчислення виразу. Давайте знову звернемося до оператора 4. Константи 5 і 9 мають цілий тип, тому результат їх розподілу також цілочисельний. Природно, що результат подальших обчислень не може бути нічим, крім нуля. Виправити цю помилку просто - достатньо записати хоча б одну з констант у вигляді дійсного числа, наприклад:

cels = 5. / 9 * (fahr – 32/;

Задача 5.2.

Напишіть програму для розрахунку за двома формулами. Заздалегідь підготуйте тестові приклади за другою формулою за допомогою калькулятора (результат обчислення за першою формулою повинен збігатися з другою). Для використання математичних функцій необхідно підключити до програми <math. h>

Приклад

Обчислити значення змінних:

$$y = \frac{\sqrt{m+x} + a^2 m}{2\sqrt{c+x}}; \qquad z = \frac{y + \cos^2 m}{\ln x^2}.$$

Як виглядає програма на екрані показано на рисунку 9.1, а результат роботи програми на рисунку 9.2.

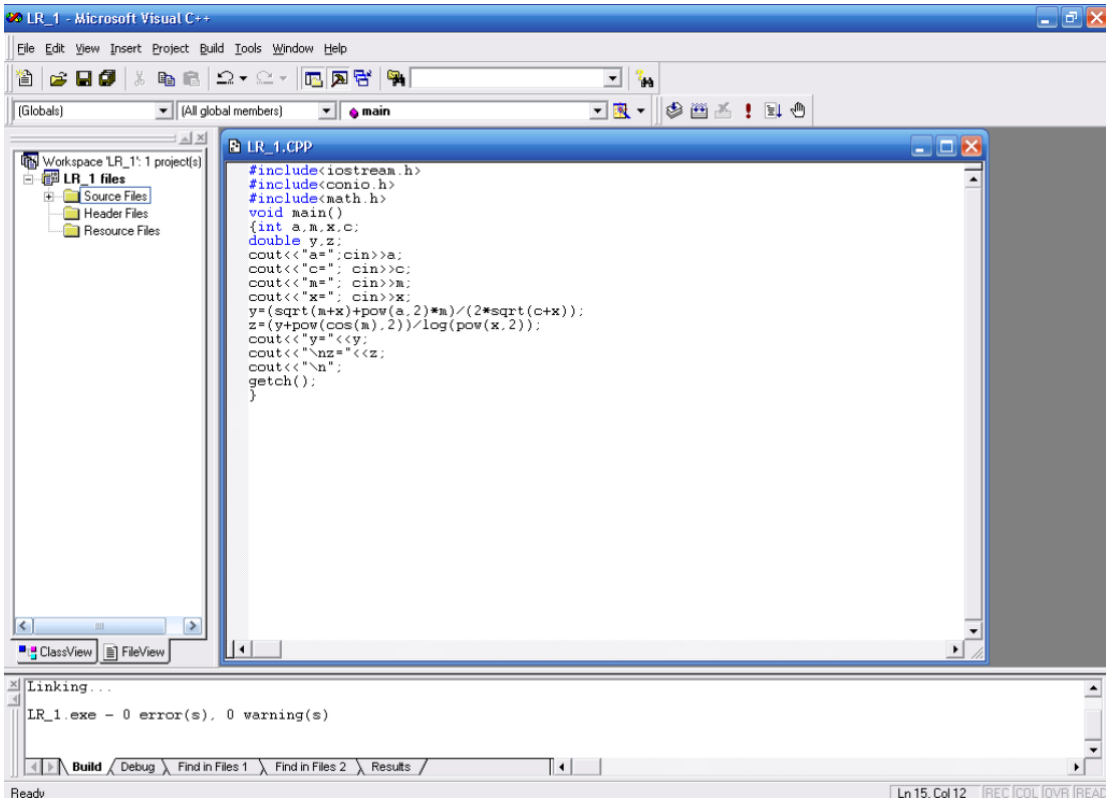


Рисунок 9.1 – Робоче вікно програми

Нижче наведений текст програми.

Лістинг програми:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>        //у цій бібліотеці знаходяться математичні
функції
void main()
{int a,m,x,c;
double y,z;
cout<<"a=";cin>>a;       //введення значення змінної a
cout<<"c="; cin>>c;
cout<<"m="; cin>>m;
cout<<"x="; cin>>x;
y=(sqrt(m+x)+pow(a,2)*m)/(2*sqrt(c+x));
z=(y+pow(cos(m),2))/log(pow(x,2));
cout<<"y="<<y;
cout<<"\nz="<<z;
cout<<"\n";
getch();
}
```

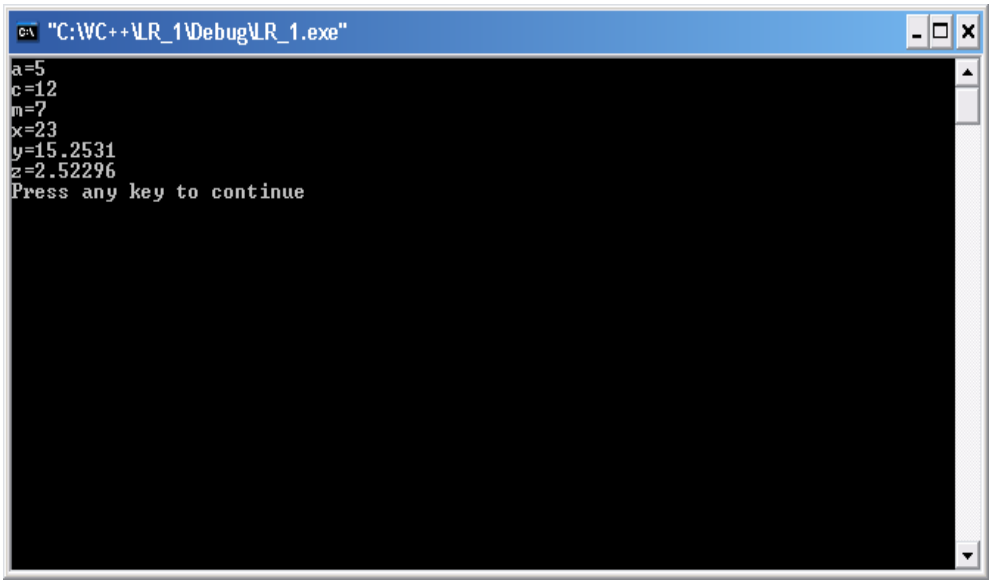


Рисунок 9.2 – Вікно результатів виконання програми

Завдання 5.1

Напишіть програму для розрахунку за двома формулами. Заздалегідь підготуйте тестові приклади за другою формулою за допомогою калькулятора (результат обчислення за першою формулою повинен збігатися з другою). Для використання математичних функцій необхідно підключити до програми <math. h>

1
$$Z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha$$
$$Z_2 = 2\sqrt{2}\cos\alpha \cdot \sin\left(\frac{\pi}{4} + 2\alpha\right)$$

2
$$z_1 = \frac{x^2 + 2x - 3 + (x+1)\sqrt{x^2 - 9}}{x^2 - 2x - 3 + (x-1)\sqrt{x^2 - 9}}$$
$$z_2 = \sqrt{\frac{x+3}{x-3}}$$

3
$$z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha$$
$$z_2 = \cos^2\alpha + \cos^4\alpha$$

4
$$z_1 = \frac{\sqrt{(3m+2)^2 - 24m}}{3\sqrt{m} - \frac{2}{\sqrt{m}}}$$
$$z_2 = -\sqrt{m}$$

5
$$z_1 = \left(\frac{a+2}{\sqrt{2a}} - \frac{a}{\sqrt{2a}+2} + \frac{2}{a-\sqrt{2a}}\right) \cdot \frac{\sqrt{a}-\sqrt{2}}{a+2}$$
$$z_2 = \frac{1}{\sqrt{a}+\sqrt{2}}$$

6
$$z_1 = \cos\alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha$$
$$z_2 = 4\cos\frac{\alpha}{2} \cdot \cos\frac{5}{2}\alpha \cdot \cos 4\alpha$$

7
$$z_1 = \cos^4 x + \sin^2 y + \frac{1}{4}\sin^2 2x - 1$$
$$z_2 = \sin(y+x) \cdot \sin(y-x)$$

8
$$z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\alpha}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\alpha}{4}\right)$$
$$z_2 = \frac{\sqrt{2}}{2}\sin\frac{\alpha}{2}$$

10
$$z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)}$$
$$z_2 = ctg\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$$

Список літератури

1 Бантюков С.Є. Використання інтегрованого середовища BORLAND C++ для розв'язання інженерно – технічних задач. Конспект лекцій з дисципліни "Комп'ютерна техніка та програмування". – Харків: УкрДАЗТ, 2006. –Ч.1, 2.

2 Павловская Т.А. C/C++ Программирование на языке высокого уровня. – С.Пб., 2003.

3 Глушаков С.В. и др. Язык программирования C++. – Харьков: Фолио, 2003.

4 Основи алгоритмізації обчислювальних процесів. Методичні вказівки до лабораторних робіт з дисципліни «Комп’ютерна техніка і організація обчислювальних робіт». – Харків, УкрДАЗТ, 2010. – № 981.

5 Методичні вказівки до лабораторних робіт з дисципліни «Основи інформатики», «Обчислювальна техніка та програмування для ЕОМ» (основи проектування алгоритмів). – Харків: ХарДАЗТ, 2002. – Ч.1 – 54.

6 Основи алгоритмізації базових обчислювальних процесів. Навч. посібник / О.В. Головко, В.С. Меркулов, В.О. Гончаров, І.Г. Бізюк, В.М. Бутенко – Харків: УкрДАЗТ, 2008. – 163 с.