**МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ**

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ"

**для студентів напряму підготовки 6.050101
"Комп'ютерні науки"
всіх форм навчання**

Харків. Вид. ХНЕУ, 2012

Затверджено на засіданні кафедри інформаційних систем.
Протокол № 2 від 14.09.2011 р.

Укладачі: Федорченко В. М.
Парфьонов Ю. Е.
Щербаков О. В.
Тарасов О. В.

М54 Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Алгоритмізація та програмування" для студентів напряму підготовки 6.050101 "Комп'ютерні науки" всіх форм навчання / укл. Федорченко В. М., Парфьонов Ю. Е., Щербаков О. В. та ін. – Х. : Вид. ХНЕУ, 2012. – 180 с. (Укр. мов.)

Подано методичні рекомендації до виконання лабораторних робіт з першого змістовного модуля даної навчальної дисципліни, що дозволять студентам закріпити теоретичні знання та продемонструвати творчий підхід до розробки проектів, грамотне використання існуючого програмного забезпечення та навички програмування на мовах високого рівня C/C++.

Рекомендовано для студентів напряму підготовки 6.050101 "Комп'ютерні науки".

Вступ

Методичні рекомендації призначені для виконання лабораторних робіт з першого модуля "Введення в розробку і кодування алгоритмів" навчальної дисципліни "Алгоритмізація та програмування".

Перед виконанням кожної роботи необхідно вивчити відповідний лекційний матеріал і звернути особливу увагу на загальні положення, передуючі опису лабораторних завдань.

Наведені приклади програм слід розглядати лише як один із можливих варіантів розв’язання завдань.

Методичні рекомендації містять опис 9 лабораторних робіт. Кожен розділ, який відповідає окремій лабораторній роботі, 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 IBM-PC з процесором не нижче Pentium III, операційна система Windows XP або Windows 7, середовище візуальної розробки програм Microsoft Visual Studio .NET.

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 вибирається відповідно до номера студента в журналі групи.

Типовий порядок виконання роботи й методичні рекомендації до її виконання:

уважно ознайомитися з методичними рекомендаціями до конкретної лабораторної роботи (теоретичними відомостями, прикладами, формулюванням завдань);

створити заготівку консольного застосування, скористатися для цього майстром створення додатків Microsoft Visual Studio;

заповнити отриману заготівку консольного застосування конкретним змістом відповідно до запропонованого завдання (див. приклади);

усунути всі помилки, що виникли на етапі компіляції початкового тексту програми;

виконання програми здійснити в покроковому режимі;

вивести у вікно попереднього перегляду значення всіх проміжних змінних;

знайти свою папку проекту і ознайомитися з її вмістом (за допомогою Блокнота відкрити файл ReadMe.txt і перекласти текст, який у ньому записаний);

підсумковий запуск додатка виконати за допомогою виконуваного модуля;

відповісти на контрольні питання;

за допомогою динамічної довідки з'ясувати призначення основних службових слів у програмі;

виконати експериментальну частину роботи згідно з отриманим завданням;

оформити звіт і здати викладачеві.

Звіт з будь-якої лабораторної роботи повинен містити:

1. Титульний лист:
   - назва дисципліни;
   - тема лабораторної роботи;
   - дата виконання роботи;
   - П.І.Б. студента, курс, номер групи;
   - П.І.Б., посада викладача.

2. Лист змісту (нумерований перелік назв пунктів роботи із зазначенням номерів сторінок).

3. Опис виконаних завдань:

Умова завдання (завдань).

Опис архітектури програми – специфікація програмних вимог (склад, структура модулів, зв'язки між ними, алгоритми):

формулювання завдання;

специфікація даних;

математична модель обробки даних;

програмний інтерфейс;

план тестування.

Початковий код програми з коментарями для бібліотек (призначення кожної бібліотеки), що підключаються, об'яв, інструкцій, що управляють, і функцій (призначення кожної функції та інструкції, опис параметрів і повертаного значення).

Приклади результатів роботи програми на тестових початкових даних.

4. Висновки за роботою з урахуванням усіх виконаних завдань:

аналіз отриманих результатів за кожним пунктом завдання;

аналіз результатів тестування програм;

ступінь відповідності розроблених програм постановці завдання;

інша інформація.

Приклад звіту наведений у додатку А. Викладач може вносити корективи до оформлення звіту.

Вимоги до оформлення звіту

Поля сторінки: ліве – 2,5 см, праве – 1,5 см, верхнє й нижнє – 2 см; шрифт Times New Roman (висота – 14 пт), міжрядковий інтервал – множник 1,1.

Номери сторінок повинні знаходитися у правому верхньому куті, титульний лист не нумерується.

Листи звіту мають бути з´єднані скріпкою або іншим загальноприйнятим способом.

Лабораторна робота № 1

Частина 1

Знайомство з інтегрованим середовищем розробки програм MS Visual C++.NET

Мета лабораторної роботи – придбання первинних практичних навиків роботи в середовищі візуальної розробки програм Microsoft Visual Studio .NET.

Перед виконанням лабораторної роботи студент повинен знати:

призначення основних елементів вікна Microsoft Visual Studio .NET;

основні органи управління вікном Microsoft Visual Studio .NET;

структуру й призначення основних елементів формату С-програми;

призначення директив препроцесора, операторів програми і коментарів;

призначення й способи використання динамічної довідки.

Після виконання лабораторної роботи студент повинен вміти:

створювати заготівку проекту;

здійснювати первинне введення й редагування тексту програми;

знаходити й усувати помилки, які виникли на етапі трансляції програми.

Розробка простого консольного додатка

Запустіть Microsoft Visual Studio 2010, для чого виберіть меню

Пуск > Програми > Microsoft Visual Studio 2010.

Після цього на екрані з'явиться головне вікно інтегрованого середовища розробки Microsoft Visual Studio 2010 (рис. 1). Детальний опис інтегрованого середовища наведений у додатку Б.

Для створення консольного додатка оберіть меню File, в ньому – команду New, а потім – команду Project (рис. 2).

З'явиться діалогове вікно New Project (рис. 3). У цьому діалоговому вікні вкажіть у розділі Project Types (Тип проекту) значення Win 32, в розділі Templates (Шаблон) – значення Win 32 Console Application (Консольний додаток), вкажіть у полі Name (Ім'я) ім'я проекту, наприклад, first (перший) і виберіть теку, в яку буде поміщений новий проект. Для цього в полі Location (Розміщення) виберіть диск і теку. Натисніть кнопку "OK" і на екрані з'явиться діалогове вікно Win 32 Application Wizard (рис. 4).

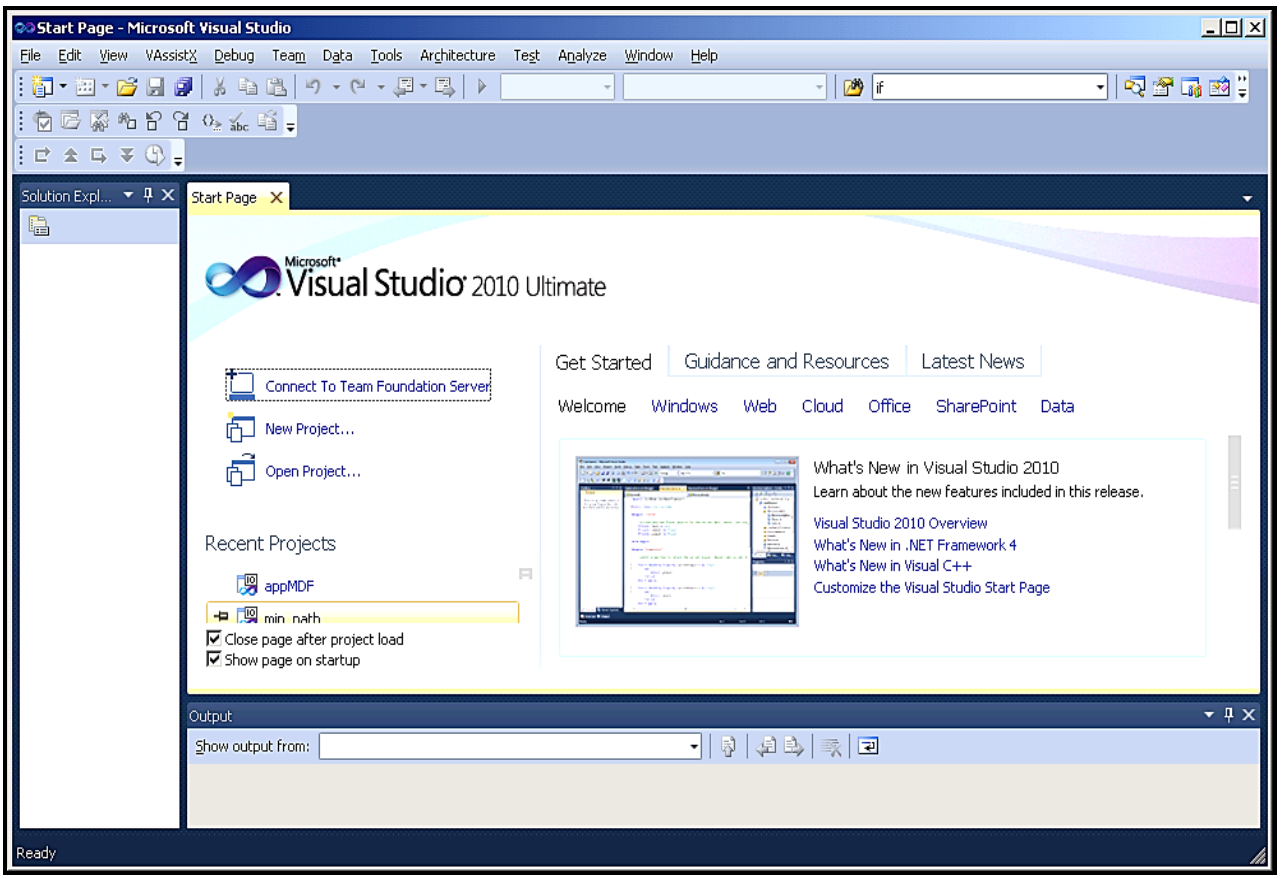


Рис. 1. **Головне вікно інтегрованого середовища розробки**

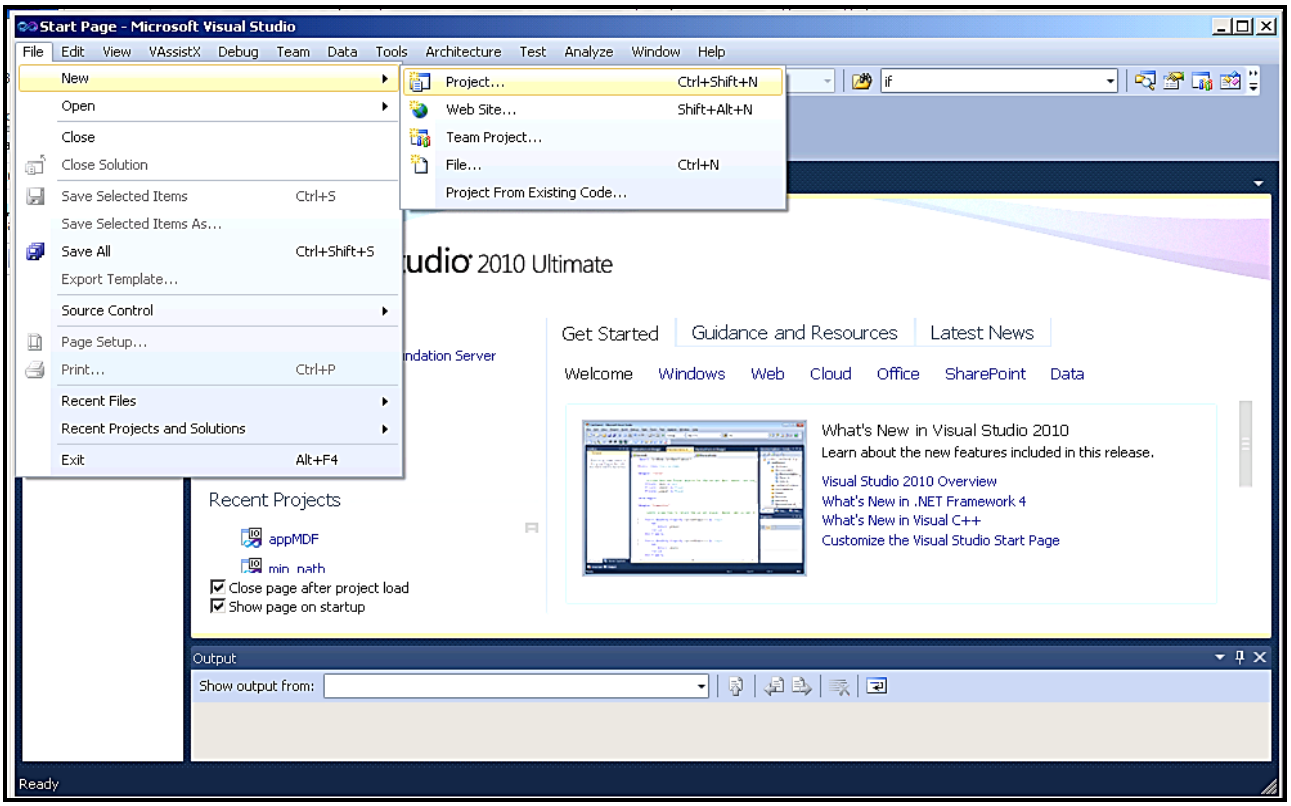


Рис. 2. **Команда Project**

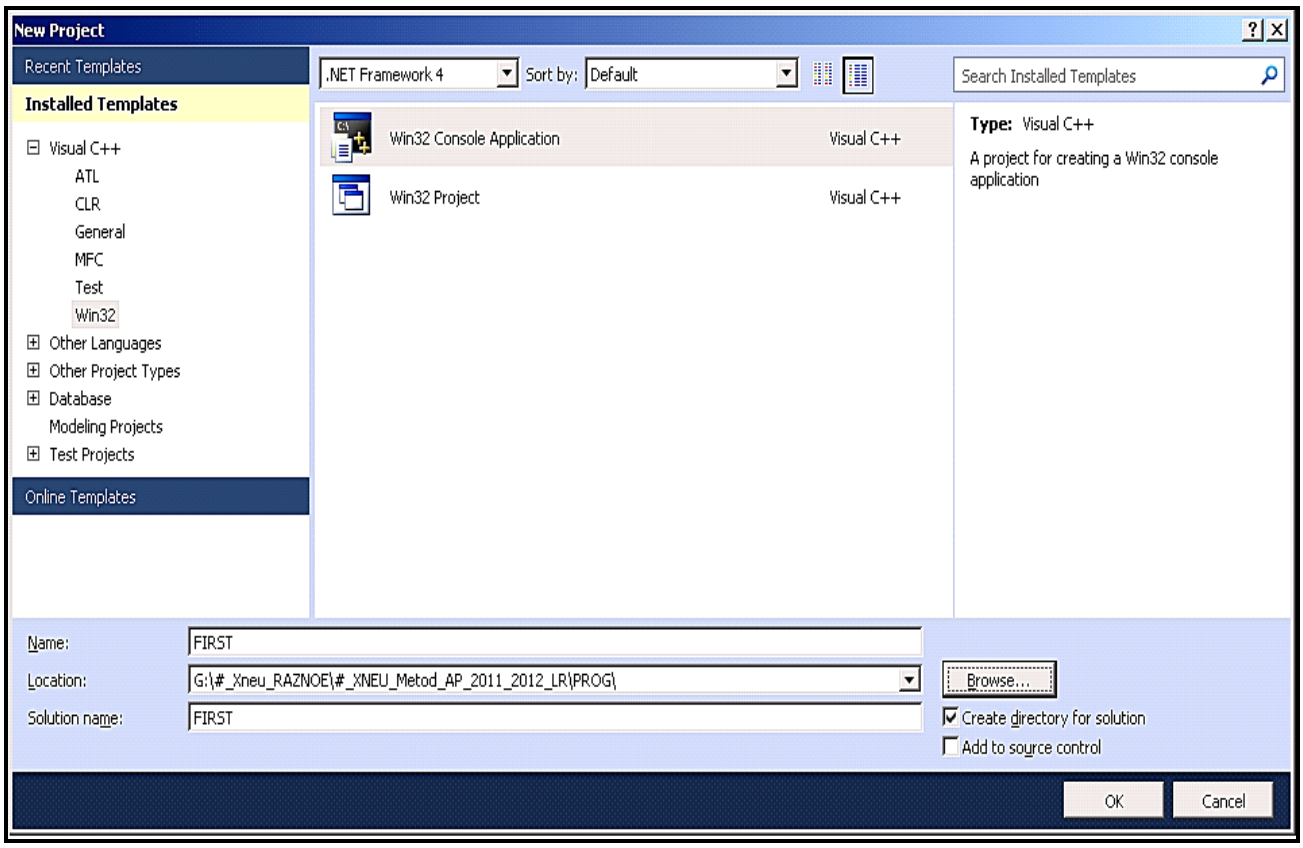


Рис. 3. **Діалогове вікно New Project**

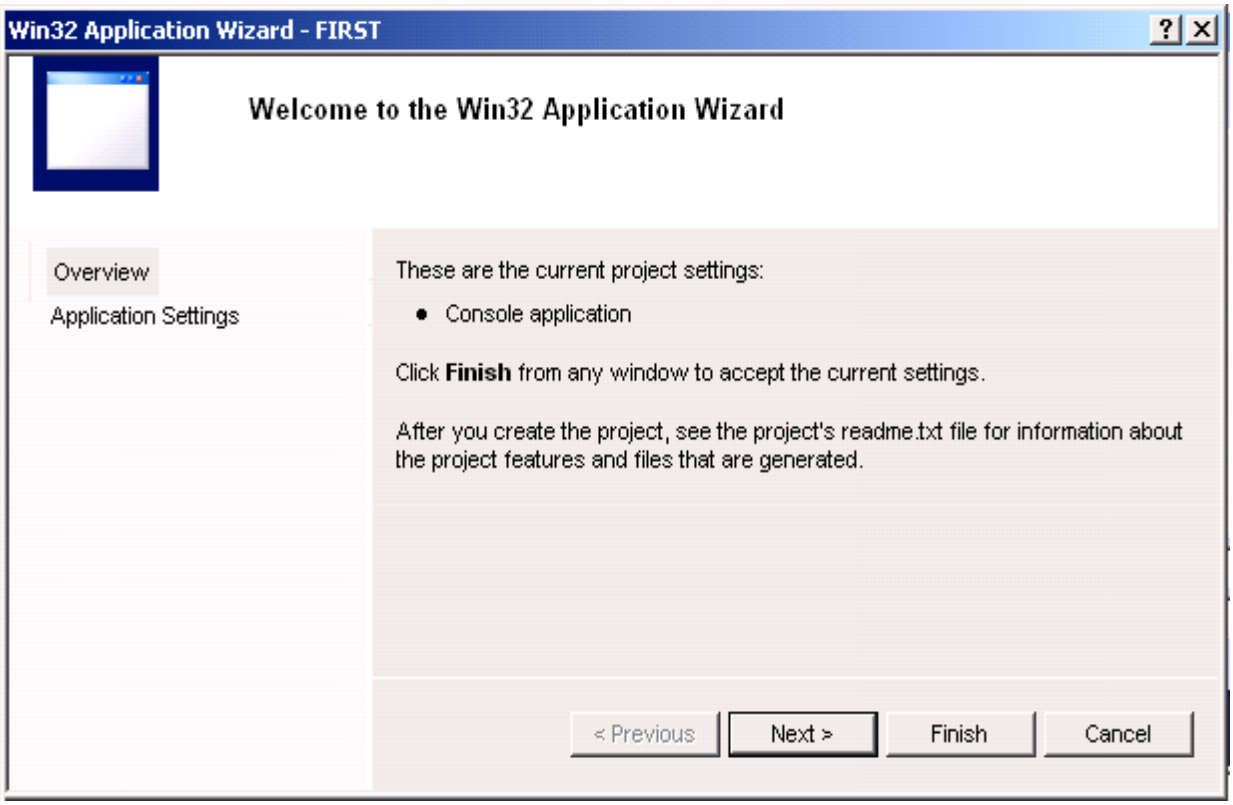


Рис. 4. **Діалогове вікно Win 32 Application Wizard**

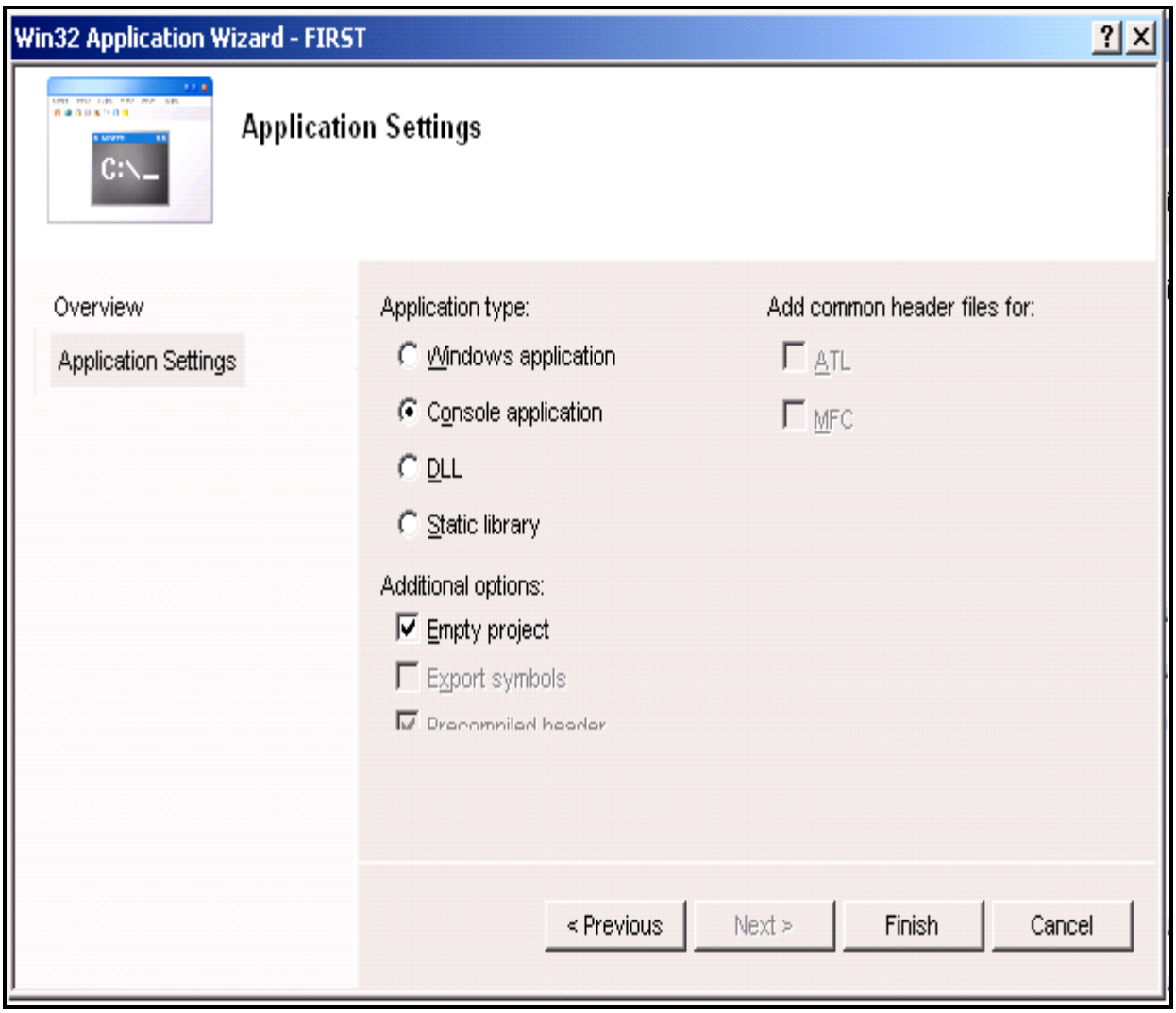


Рис. 5. **Вікно – Application Settings (Установки додатку)**

У діалоговому вікні Win 32 Application Wizard натисніть кнопку Next і на екрані з'явиться наступна сторінка цього вікна – Application Settings (Установки додатка) (рис. 5). У групі перемикачів Application type (Тип додатка) виберіть Console Application (Консольний додаток), а в групі Additional options (Додаткові опції) виберіть Empty project (Порожній проект) (рис. 5). Після цього натисніть кнопку Finish.

На екрані знову з'явиться головне вікно інтегрованого середовища розробки Microsoft Visual Studio 2010, в лівій частині якого, у вікні провідника рішення (Solution Explorer) з'являться папки нашого проекту (рис. 6)

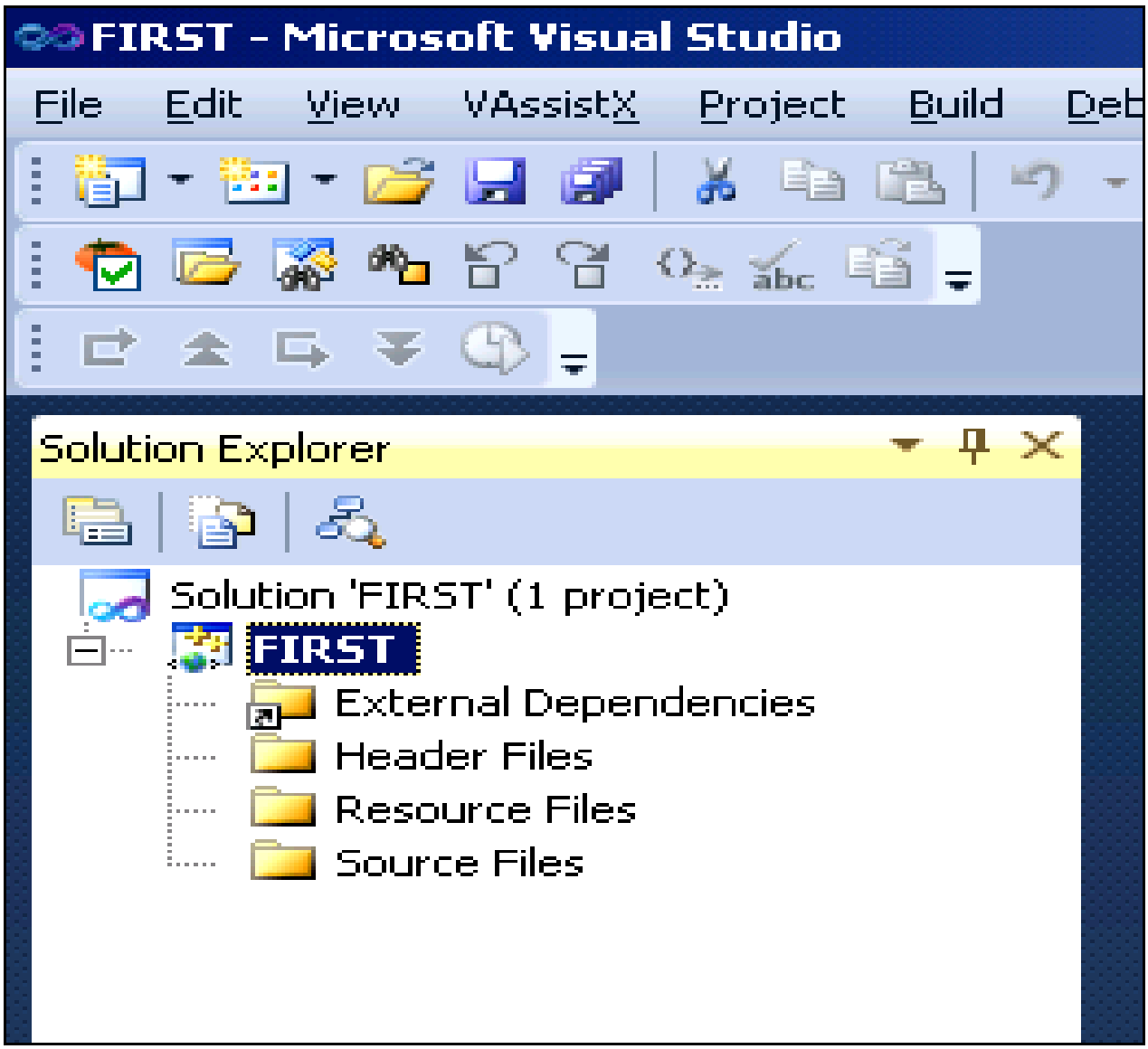


Рис. 6. **Вікно провідника рішення (Solution Explorer)**

Клацніть правою кнопкою миші на папці Source Files у вікні провідника рішення (Solution Explorer). У контекстному меню вікна, що з'явилося, виберіть команду Add і далі в додатковому меню, що з'явилося, виберіть команду Add New Item. У результаті на екрані з'явиться діалогове вікно Add New Item (рис. 7).

У цьому вікні виберіть у розділі Categories пункт Code, в розділі Templates – пункт C++File(.cpp) і вкажіть у полі Name ім'я файлу, наприклад, prog1 (рис. 7). Буде створений порожній файл prog1.cpp і відкриється редактор коду програми для введення тексту програми (рис. 8).

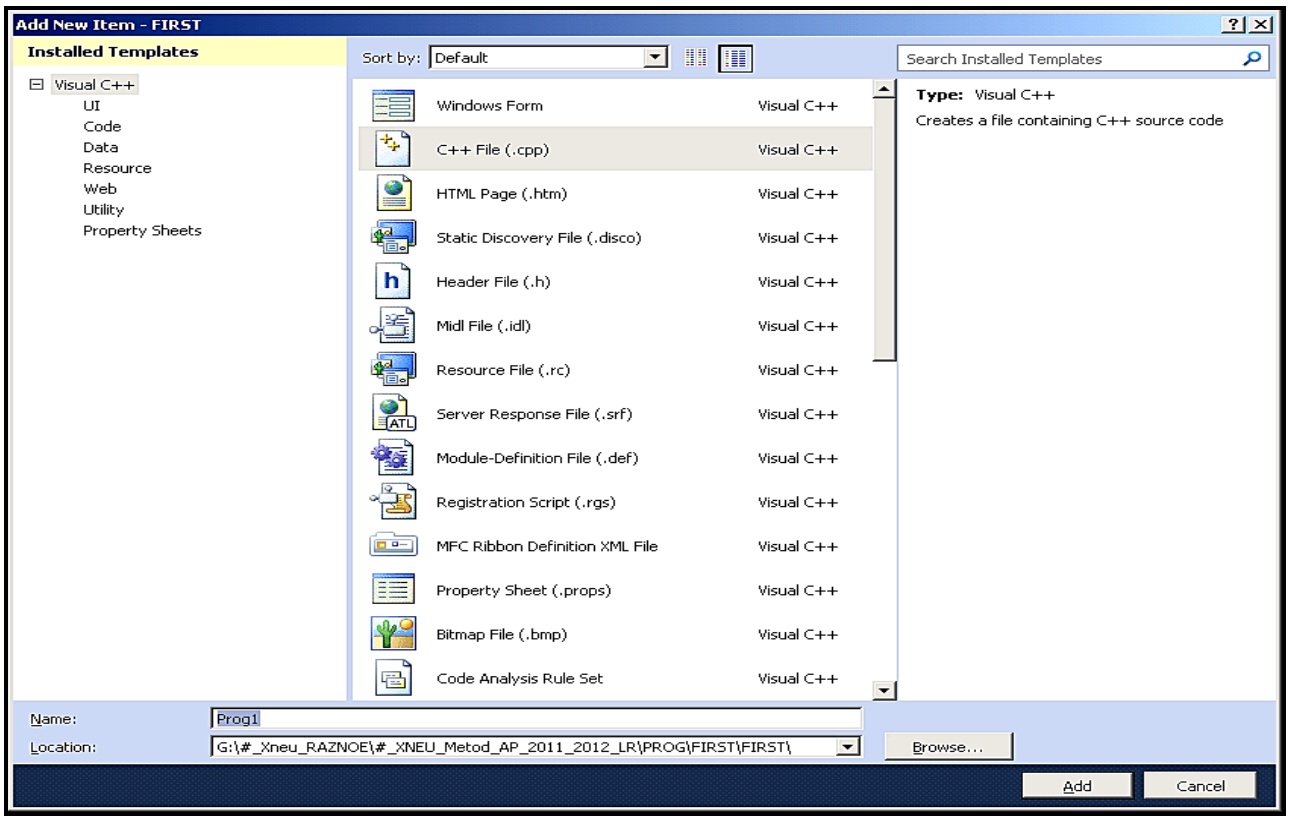


Рис. 7. **Діалогове вікно Add New Item**

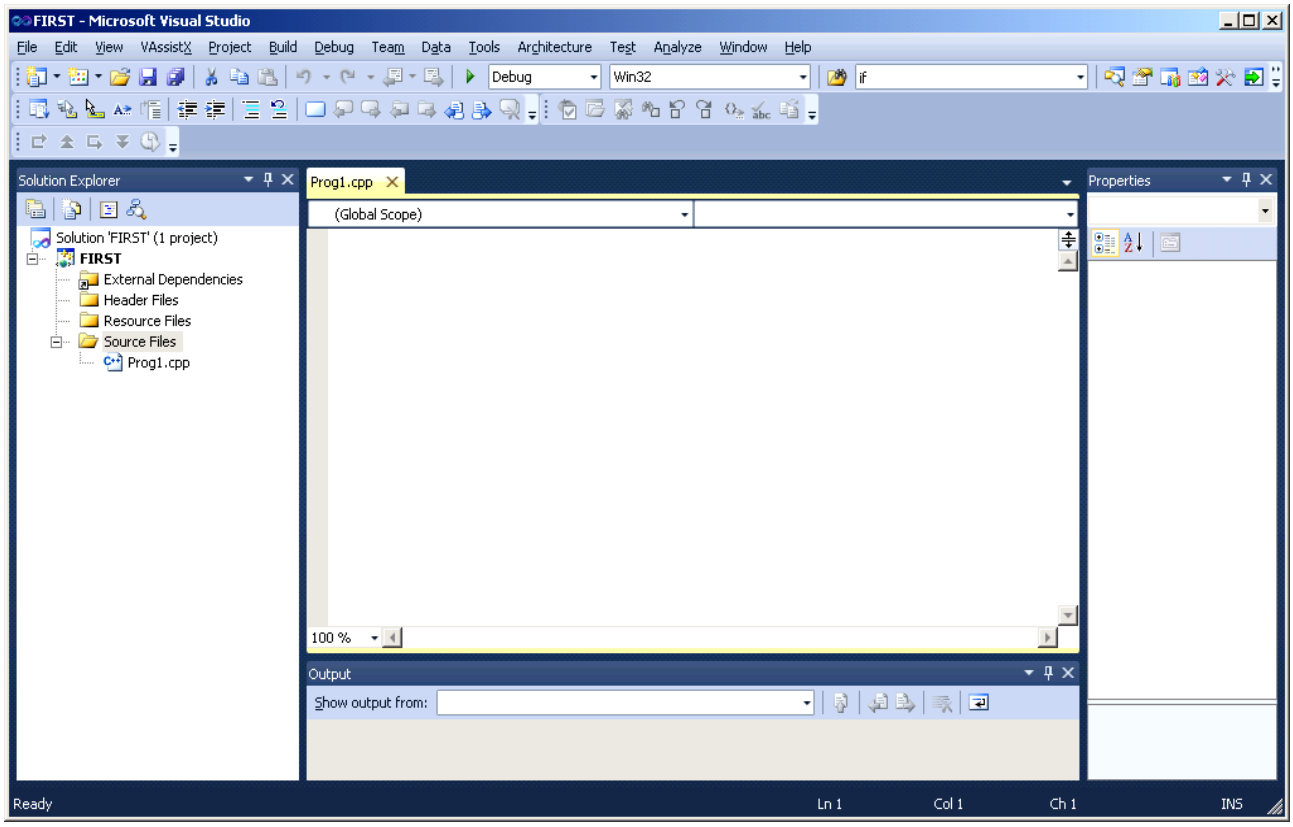


Рис. 8. **Редактор коду програми**

Введіть у редакторові коду такий текст:

```
#include <iostream>
int main()
{
    std::cout << "Hello!";
    getchar();
    return 0;
}
```

Вікно повинне набути такого вигляду (рис. 9):

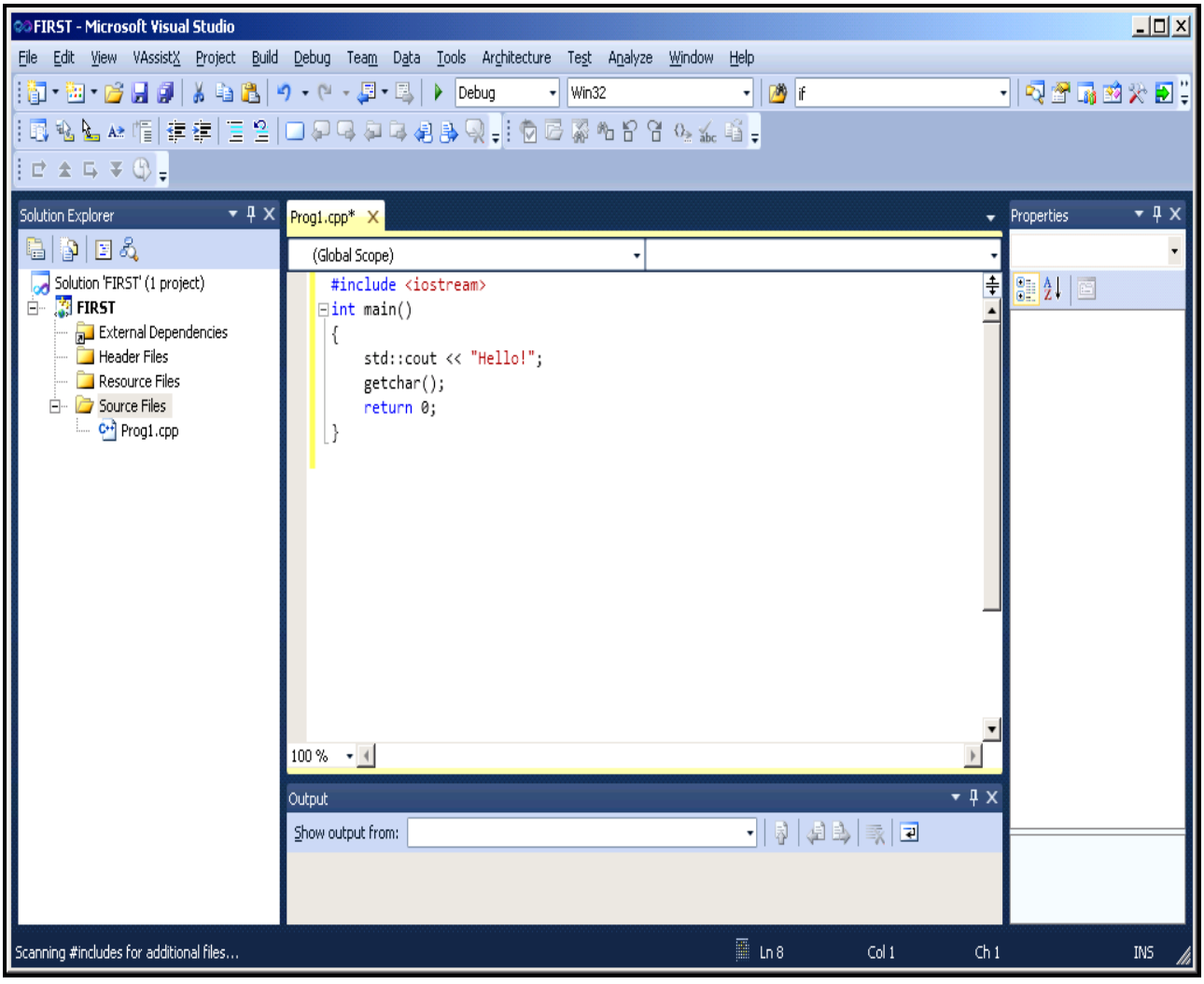


Рис. 9. **Вікно з кодом програми**

Збережіть початковий текст програми на диску, вибравши пункт меню File, а в ньому – команду Save All.

Відкомпілюйте програму й створіть виконуваний файл. Для створення виконуваного файлу рекомендується використовувати команду меню Build> Rebuild Solution.

Натиснувши функціональну клавішу F5 або кнопку Start Debugging на панелі інструментів, запустите додаток на виконання. У результаті ви побачите консольне вікно з виведеним рядком вітання Hello! (рис. 10).

Рис. 10. **Консольне вікно**

Натисніть клавішу Enter, щоб повернутися в головне вікно інтегрованого середовища розробки Microsoft Visual Studio 2010.

Виберіть пункт меню File, а в ньому – команду Exit, щоб завершити роботу з програмою Microsoft Visual Studio 2010.

Порядок виконання роботи й методичні рекомендації до її виконання:

створити на робочому диску папку для проектів (ім'я папки повинне містити вичерпну інформацію про її користувача);

запустити додаток Visual Studio.Net;

ознайомитися з елементами вікна Visual Studio.Net;

ознайомитися з командами кожного пункту головного меню;

ознайомитися з елементами управління вікном Visual Studio.Net (звернути увагу на роботу з ковзаючими вікнами);

отримати у викладача навчально-демонстраційну програму або скористатися програмою, яка використовується в контексті опису середовища;

виконати дії з введення, редагування, відладки й побудови виконуваного модуля навчально-демонстраційної програми;

знайти свою теку проекту й ознайомитися з її вмістом (за допомогою Блокнота відкрити файл ReadMe.txt і перекласти текст, який в ньому записаний);

підсумковий запуск виконуваного модуля виконати з командного рядка;

змінити властивості консольного вікна й зробити фон вікна білим, а шрифт чорним;

за допомогою динамічної довідки з'ясувати призначення всіх службових слів у програмі (службові слова в програмі забарвлені блакитним кольором).

Контрольні запитання

1. Назвіть основні етапи розробки програми на ПЕОМ.
2. Назвіть основні елементи вікна додатка Visual Studio.Net.
3. Назвіть основні органи управління вікном додатка Visual Studio.Net.
4. Перерахуйте засоби відладки програм у середовищі Visual Studio.Net.
5. Назвіть основні елементи структури С-програми.
6. Перерахуйте основні операції з редагування тексту програми.
7. Чим відрізняється оператор мови програмування С++ від коментаря?
8. Яке завдання вирішується на етапі компіляції початкового тексту програми?
9. Яке завдання вирішується на етапі побудови виконуваного модуля програми?

Лабораторна робота № 1

Частина 2

Підготовка і розв’язання на ПЕОМ задач лінійного характеру

Мета лабораторної роботи – придбання практичних навиків з підготовки, відладки та виконання лінійних програм.

Змістовний сенс практичного завдання – обчислення похідної в точці, обчислення значення виразу.

Перед виконанням лабораторної роботи студент повинен знати:

класифікацію базових типів даних і 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 відладку й покрокове тестування лінійних програм.

Короткі теоретичні відомості

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

Для програмування завдань лінійного характеру (рис. 11), 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.

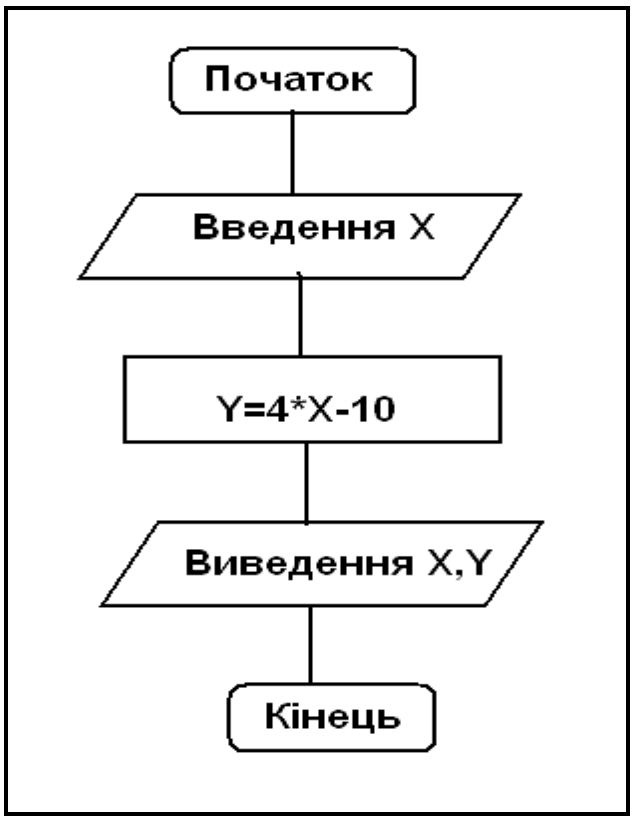


Рис. 11. **Лінійний алгоритм**

Вирази в мові С++ записуються за допомогою 26 рядкових букв англійського алфавіту:

abcdefghijklmnopqrstuvwxyz ;

26 прописних букв англійського алфавіту:

ABCDEFGHIJKLMNOPQRSTUVWXYZ ;

десяти цифр:

0123456789 ;

таких спеціальних символів:

+ - * / =, . _ : ; ? \ " ’~ | ! # $ & ( ) [ ] { } ^ @.

До спеціальних символів відноситься також пропуск. Комбінації деяких символів, не розділених пропусками, інтерпретуються як один значущий символ:

++ -- || << >> >= <= += -= *= /= .?: :: /* */ //.

Ідентифікаторами називаються імена, що привласнюються змінним, константам, типам даних і функціям, які використовуються в програмах. Після опису ідентифікатора можна посилатися на об'єкт, що позначається ним, у будь-якому місці програми.

Ідентифікатор є послідовністю символів довільної довжини, що містить букви, цифри й символи підкреслення, яка обов'язково повинна починатися з букви або символу підкреслення.

У С++ враховується регістр букв. Компілятор сприймає прописні і рядкові букви, як різні символи. Так, змінні NM_LEN і Nm_Len розглядаються як два разні ідентифікатори.

Використання символу підкреслення на початку імені ідентифікатора не рекомендується, оскільки даний спосіб запису застосовується в іменах системних підпрограм і змінних. Збіг імені ідентифікатора із зарезервованим ім'ям викличе конфлікт у роботі програми. Два символи підкреслення (__) на початку імені ідентифікатора застосовуються в стандартних бібліотеках мови С++.

У процесі формування імен змінних і функцій прийнята угода починати їх з префікса типу даних цього ідентифікатора (наприклад, ідентифікатор ia відповідає типу int), ідентифікатори з плаваючою комою – буквою f (float), рядки, що завершуються нульовим символом, – буквами sz (string zero), покажчики – буквою p (pointer) і т. д. Це істотно спрощує сприйняття текстів програм.

Ключові слова є зарезервованими ідентифікаторами, кожному з яких відповідає певна дія. Змінити призначення ключового слова неможна (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++ (табл. 1).

Таблиця 1

Ключові слова C/C++

__alignof	__stdcall	else	return
__asm	__super	enum	short
__assume	__try __except	explicit	signed
__based	__try __finally	extern	sizeof
__cdecl	__unaligned	false	static
__declspec	__uuidof	float	static_cast
__alignof	__virtual_inheritance	for	struct
__except	auto	friend	switch
__fastcall	__unaligned	goto	template
__finally	bool	extern	this
__forceinline	break	if	throw
__inline	case	inline	true
__int16	catch	int	try
__int32	char	long	typedef
__int64	class	mutable	typeid
__int8	const	namespace	typename
__interface	const_cast	new	union
__leave	continue	operator	using
__multiple_inheritance	default	private	unsigned
__noop	delete	protected	virtual
__pragma	do	public	void
__ptr64	double	register	volatile
__sealed	dynamic_cast	reinterpret_cast	wchar_t
__single_inheritance			while

Ключові слова, що починаються із знаків підкреслення, визначені компанією Microsoft.

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

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

Текстом (тип даних char) є послідовність символів, які можуть бути розділені пропусками. 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 C++ підтримуються три типи цілих чисел. Разом із стандартним типом int існують типи short int (коротке ціле) і long int (довге ціле). Допускається скорочений запис short і long.

Числа з плаваючою комою 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 біт, 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 додатками.

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 C++ можуть містити тільки одну з двох констант: true або false.

Компілятор мови C++ дозволяє при описі 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 знаку. Існує модифікатор signed, який виконує протилежну unsigned дію.

У C++ 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 потрібне, 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). У бухгалтерських програмах такою величиною є ПДВ.

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.

Допускається в деяких випадках одночасне використання кваліфікаторів const і volatile.

У C++ застосовуються й інші кваліфікатори. Їх розгляд виходить за рамки методичних рекомендацій.

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

```
int ival;
float fres;
ival=ival*fres;
```

У процесі виконання змішаних операцій компілятор автоматично проводить перетворення типів даних. Цілочисельне значення змінної ival зчитується з пам'яті, приводиться до типу з плаваючою комою й помножується на початкове значення змінної fres. Результат у вигляді значення з плаваючою комою привласнюється змінною цілого типу ival. 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 велику розмірність, немає втрати 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:

```
fr=fv+(float) iv/iw;
fr=fv+iv/(float) iw;
fr=fv+(float) iv/(float) iw;
```

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

1.3. Специфікатори класу пам'яті

У C++ є чотири специфікатори класу пам'яті: auto, register, static, extern.

Специфікатор класу пам'яті може передувати оголошенням змінних і функцій, вказуючи компілятору, як слід зберігати змінні в пам'яті і як діставати доступ до змінних або функцій. Змінні, оголошені із специфікаторами auto і register, є локальними, а із специфікаторами static і extern – глобальними. Пам'ять для локальної змінної виділяється кожного разу, досягнувши блоку, в якому оголошена змінна, і звільняється після закінчення виконання блоку. Пам'ять для глобальної змінної виділяється один раз при запуску програми й звільняється, коли робота програми закінчена.

Вказані чотири специфікатори визначають також зону видимості змінних і функцій, тобто частину програми, в межах якої до ідентифікатора можна звернутися по імені. На зону видимості змінної й функції впливає місце її оголошення в програмі. Якщо оголошення розташоване поза функцією, рівень оголошення є зовнішнім, якщо ж воно знаходиться в тілі функції – внутрішнім.

Специфікатори класу пам'яті розрізняються за сенсом в залежності від того, що оголошується – змінна або функція, а також від того, на якому рівні, зовнішньому або внутрішньому, оголошується даний ідентифікатор.

Змінна, оголошена на зовнішньому рівні, є глобальною і за замовчуванням має клас пам'яті extern. Зовнішнє оголошення може включати ініціалізацію (явну або неявну) або просто бути посиланням на змінну, що ініціалізувалася в іншому місці програми. Наприклад:

```
static int iv;        // за замовчуванням неявно присвоюється 0
static int iv=10;     // явне присвоюється
int ir=20;            // явне присвоюється
```

Зона видимості глобальної змінної розповсюджується до кінця програми. Звернення до змінної не може знаходитися вище за той рядок, в якому вона оголошена.

Змінна оголошується на зовнішньому рівні тільки один раз. Якщо в одному з файлів створена змінна з класом пам'яті static, то вона може бути оголошена під тим же ім'ям з тим же специфікатором static в будь-якому іншому початковому файлі. Оскільки статичні змінні доступні тільки в межах свого файлу, конфліктів імен не виникне.

За допомогою специфікатора extern можна оголосити змінну, яка буде доступна з будь-якого місця програми. Це може бути посилання на змінну, описану в іншому файлі або нижче в тому ж файлі. Остання особливість робить можливим розміщення посилань на змінну до її ініціалізації.

1.4. Операції

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

Побітові операції працюють із змінними як із наборами бітів, а не як із числами. Ці операції використовуються в тих випадках, коли необхідно дістати доступ до окремих біт даних (при виведенні графічних зображень

на екран). Побітові операції застосовуються тільки до цілочисельних значень. На відміну від логічних операцій, з їх допомогою порівнюються не два числа цілком, а окремі їх біти. Основні побітові операції: І (&), АБО (|) і що виключає АБО (^). Сюди можна також зарахувати унарну операцію побітового заперечення (~), яка інвертує значення бітів числа.

Операція & записує в біт результату одиницю тільки в тому випадку, якщо обидва порівнюваних біта дорівнюють 1, як показано в такій таблиці:

Біт 0	Біт 1	Результат
0	0	0
0	1	0
1	0	0
1	1	1

Ця операція часто використовується для маскування окремих бітів числа. Наприклад: 0xF1 & 0x 35 = 0x 31.

Операція | записує в біт результату одиницю в тому випадку, якщо хоч би один з порівнюваних бітів дорівнює 1, як показано в такій таблиці:

Біт 0	Біт 1	Результат
0	0	0
0	1	1
1	0	1
1	1	1

Ця операція часто застосовується для установки окремих бітів числа. Наприклад: 0x F1 | 0x 35 = 0x F5.

Операція ^ записує в біт результату одиницю в тому випадку, якщо порівнювані біти відрізняються один від одного, як показано в такій таблиці:

Біт 0	Біт 1	Результат
0	0	0
0	1	1
1	0	1
1	1	0

Ця операція часто застосовується при виведенні зображень на екран, коли відбувається накладення декількох графічних шарів.

Наприклад: 0x F1 ^ 0x 35 = 0x C4.

У С++ існує дві операції зсуву: << - зсув ліворуч, >> - зсув праворуч. Дія першої операції полягає в 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 еквівалентне множенню числа на 2. Наприклад:

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

Зсув праворуч супроводжується аналогічними діями, тільки бітове представлення числа зрушується на вказану кількість бітів управо. 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 iv=10;     // молодший байт: 00001010
iv>>=1;                 // молодший байт: 00000101
cout<<iv;               // 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 iv+1, можна ввести рядок iv++ або ++iv.

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 з 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 і++ спочатку повертається значення змінної, після чого воно збільшується на одиницю. З іншого боку, операція префіксного інкрементування ++і вказує, що спочатку слід збільшити значення змінної, а потім повернути його як результат. Наприклад:

```
k=++ i;  // i=4, k=4
k=i++;   // i=4, k=3
k=--i;   // i=2, k=2
k=i--;   // i=2, k=3
```

У C++ 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 ia=3, ib=8, id;  id=ib % ia;    // результат: 2.
```

При діленні по модулю повертається залишок від операції цілочисельного ділення.

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

```
iv=8*(iw=5); //iv=40.
```

У даному випадку спочатку змінній iw присвоюється значення 5, після чого це значення помножується на 8, а результат присвоюється змінній iv. Комбіновані операції присвоєння наведені у табл. 2.

Таблиця 2

Комбіновані операції привласнення

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 3 і C++.

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 помилкові (дорівнюють нулю). Оператор ! інвертує значення свого операнда з false на frue і навпаки.

Приклад використання логічних операцій і операцій порівняння наведений нижче:

```
#include <iostream>
#include <conio.h>
using namespace std;
void main() {
float fa=2,fb=4;
cout<<"fa<fb  " <<(fa<fb) <<"\n";
cout<<"fa>fb  " <<(fa>fb) <<"\n";
cout<<"fa<=fb " <<(fa<=fb) <<"\n";
cout<<"fa>=fb  " <<(fa>=fb) <<"\n";
cout<<"fa==fb  " <<(fa==fb) <<"\n";
cout<<"fa!=fb   " <<(fa!=fb) <<"\n";
cout<<"fa&&fb  " <<(fa&&fb) <<"\n";
cout<<"fa||fb  " <<(fa||fb) <<"\n";
getch();
}
```

Результат роботи програми наведений на рис. 12

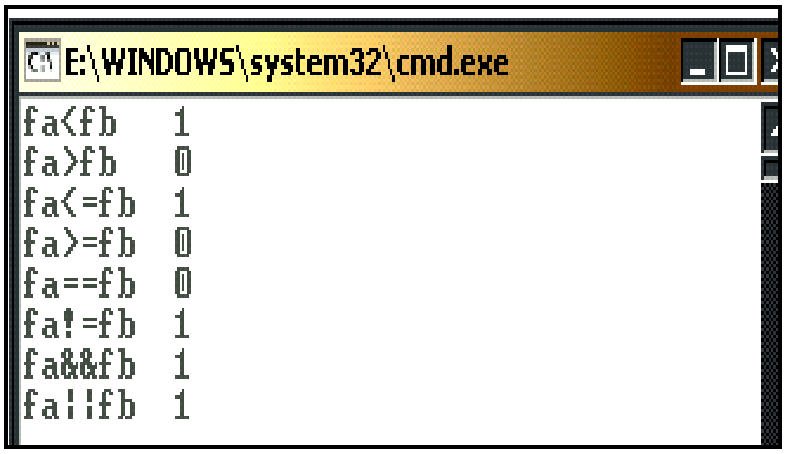


Рис. 12. **Результат роботи програми**

Умовна операція має такий формат:

Умовний вираз ? вираз 1 : вираз 2;

Якщо умовний вираз **true**, то виконується вираз 1. Якщо умова **false**, то виконується вираз 2. Наприклад:

var>7 ? x=11:y=7;

Часто вирази 1 і 2 використовують одну і ту ж змінну, якою привласнюється або одне, або інше значення. Тоді умовна операція записується трохи інакше:

Змінна = Умовний вираз ? вираз 1 : вираз 2;

Наприклад:

char cS=(x<=max) ? 'Y ' :'N';

Якщо x<=max, то змінна cS набуває значення 'Y', якщо x>max, то cS буде рівна 'N'.

Операція кома (,) дозволяє послідовно виконати два вирази, записані в одному рядку. Результатом є значення виразу, розташованого праворуч від коми. Синтаксис оператора такий:

лівий_вираз, правий_вираз

Найчастіше цей оператор застосовується в циклі for, коли в умову циклу потрібно включити перевірку значень декількох змінних. Наприклад:

for(min=0,max=len-1;min<max;min++,max--) { . }

Операція sizeof служить для визначення розміру операнда в байтах. Вона може використовуватися як з позначенням змінної, так і з її

типом (у останньому випадку операнд слід укласти в круглі дужки). Операція sizeof особливо корисна для визначення розмірів агрегатних змінних – масивів і структур. Наприклад:

```
char cS[]="Операция sizeof";
int cSLen=sizeof cS;
```

Масив cS займатиме 16 байтів пам'яті.

Послідовність виконання різних операцій визначається компілятором.

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

У табл. 3 перераховані всі операції мови C++ в порядку зниження їх пріоритету і вказаний напрям обчислення операндів (асоціативність): зліва направо або справа наліво.

Таблиця 3

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	
*	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
->*	Непрямий доступ до покажчика на член класу (через покажчик на об'єкт)	

Закінчення табл. 3

1	2	3
*	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 привласнення	Справа наліво
*=	Привласнення з множенням	
/=	Привласнення з діленням	
%=	Привласнення з діленням по модулю	
+=	Привласнення зі складанням	
-=	Привласнення з відніманням	
<<=	Привласнення із зсувом ліворуч	
>>=	Привласнення із зсувом управо	
&=	Привласнення з побітовим І	
|=	Привласнення з побітовим АБО	
^=	Привласнення з побітовим виключним АБО	
,	Кома	Зліва направо

1.5. Стандартні бібліопапки C++ і бібліотечні математичні функції

Різним бібліотечним функціям потрібні різні заголовні файли. Заголовні файли, які необхідні функції, вказуються в її описі. Наприклад, функції sqrt() потрібні оголошення, що містяться в заголовному файлі

math.h. У Microsoft Visual C++ Run-Time Library Reference перераховані всі бібліотечні функції й відповідні заголовні файли.

Розробниками компілятора C++ передбачені такі категорії бібліотек: класифікації, перетворення, управління каталогами, діагностики, програми, введення/виводу, інтерфейсні, обробки, математичні, управління пам'яттю, управління процесом, стандартні, виведення текстових вікон, для обробки інформації про час і дату.

Стандартні бібліотечні математичні функції:

int abs(int x);

double fabs(double x);

Повертає ціле (abs) або дробове (fabs) абсолютне значення аргументу, який можна використовувати вираз відповідного типу.

double acos (double x);

double asin (double x);

double atan (double x);

long double acosl(long double x);

long double asinl(long double x);

long double atanl(long double x);

Повертає виражену в радіанах величину кута, косинус, синус або тангенс якого переданий відповідній функції як аргумент. Аргумент функції повинен знаходитися в діапазоні від -1 до 1.

double cos (double x);

double sin (double x);

double tan (double x);

long double cosl(long double x);

long double sinl(long double x);

long double tanl(long double x);

Повертає синус, косинус або тангенс кута. Величина кута має бути задана в радіанах.

double exp(double x);

long double exp(long double (x));

Повертає значення, яке дорівнює експоненті аргументу (e^x, де e – основа натурального логарифма).

double pow (double x, double y);

long double powl(long double (x), long double (y));

Повертає значення, яке дорівнює x^y.

double sqrt(double x);

Повертає значення, яке дорівнює квадратному кореню з аргументу.

Заголовний файл: <math.h>

int rand(void);

Повертає випадкове ціле число в діапазоні від 0 до RAND_MAX. Перед першим зверненням до функції rand необхідно ініціалізувати генератор випадкових чисел. Для цього треба викликати функцію srand.

void srand(unsigned x);

Ініціалізував генератор випадкових чисел. Зазвичай як параметр функції використовують змінну, значення якої передбачити заздалегідь не можна, наприклад це може бути поточний час.

Заголовний файл: <stdlib.h>

Наведені нижче функції виконують перетворення рядків у числове значення і чисел у строкове уявлення.

double atof(const char* s);

Повертає дробове число, значення якого передане функції як аргумент. Функція обробляє рядок до тих пір, поки символи рядка є допустимими. Рядок може бути значенням числа як у форматі з плаваючою точкою, так і в експоненціальному форматі.

int atoi(const char* s);

long atol(const char* s);

Повертає ціле відповідного типу, зображення якого передане функції як аргумент. Функція обробляє символи рядка до тих пір, поки не зустріне символ, що не є десятковою цифрою.

char *gcvt(double Значення, int Цифр, char* Рядок);

Перетворить дробове число в рядок. При перетворенні робиться спроба отримати вказану кількість значущих цифр, а якщо це зробити неможливо, то число зображується у формі з плаваючою точкою.

char* itoa (int Значення, char* Рядок, int Основа);

char* ltoa (long Значення, char* Рядок, int Основа);

char* ultoa(unsigned long Значення, char* Рядок, int Основа);

Відповідно перетворять ціле, довге ціле і довге беззнакове ціле в рядок. Число зображується у вказаній при виклику функції системі числення.

Рядок – покажчик на рядок, куди буде поміщено зображення числа. Основа – задає основу системи числення (від 2 до 36).

Максимальна довжина рядка, формованою функцією itoa – 17 байт, функціями ltoa і ultoa – 33 байти.

Заголовний файл: <stdlib.h>

int sprintf(char *Рядок, const char* Формат, Список змінних);

Виконує форматований вивід у рядок.

Спісок змінних – розділені комами імена змінних, задає змінні, значення яких мають бути виведені. Параметр Формат задає спосіб відображення значень змінних.

Дія функції sprintf аналогічно дії функції printf, але вивід виконується в рядок-буфер, а не на екран.

Заголовний файл: <stdio.h>

Бібліотечні функції введення-виводу:

int printf (Формат, Список змінних);

Виводить на екран значення змінних. Формат виводу задається в рядку форматування, який окрім специфікатора формату може містити текст і символи, що управляють. Значення першої змінної виводиться відповідно до першого специфікатора формату, другий – до другого, третій – до третього, і т. д.

Специфікатори формату (необов'язковий параметр n задає ширину поля виводу).

Специфікатор	**Форма виводу**
%ni %nd	Десяткове число із знаком
%nu	Беззнакове ціле десяткове число
%n.mf	Дробове число з десятковою точкою. Необов'язковий параметр m задає кількість цифр дробової частини
%ne	Дробове число з десятковою точкою або, якщо число не може бути представлене у формі з десятковою точкою, в експоненціальній формі
%ns	Рядок символів
%nc	Символ

Символи, що управляють, і спеціальні.

Символ	**Дія**
\n	Переводить курсор в початок наступного рядка
\t	Переводить курсор в чергову позицію табуляції
\\	Бекслеш
\’	Лапка

int scanf(const char* Формат, Список адрес змінних);

Вводить з клавіатури значення змінних, відповідно до вказаного специфікатора формату. Перша змінна набуває значення відповідно до першого специфікатора формату, друга – до другого, і т. д.

Специфікатор	Вводить
%i %d	Десяткове число із знаком
%u	Беззнакове ціле десяткове число
%e %f	Дробове число
%s	Рядок символів
%c	Символ

puts(const char* Рядок);

Виводить на екран рядок символів і переводить курсор в початок наступного рядка екрану. Як параметр функції можна використовувати строкову константу або строкову змінну.

Заголовний файл: <stdio.h>

int putch(int c);

Виводить на екран символ.

Заголовний файл: <conio.h>

int getch(void);

Повертає код символу натиснутої клавіші. Якщо натиснута службова клавіша, то функція getch повертає 0. У цьому випадку, для того, щоб визначити, яка службова клавіша натиснута, потрібно звернутися до функції getch ще раз.

Заголовний файл: < conio.h >

Приклад: Знаходження значення похідної функції в точці.

Постановка завдання

Задання функція . Знайти її похідну в точці x= π /2.

Для знаходження похідної в точці використовується відомий вираз:

$$\frac{d}{dx} f(x) = \frac{\left(f(x+dx) - f(x)\right)}{dx}$$

Крім того, оскільки π/2 ≈ 1,57, як значення x вибираємо 1,57.

Текст програми

```
#include <math.h>
#include <iostream>
int main()
{
   double f1,f2,pf,x,dx;
   dx=1.0e-11;   // Вибираємо приріст аргументу
   x=1.57;      // Вибираємо точку для обчислення похідної
   f1=sin(x+dx);  // Обчислюване значення функції в точці x+dx
   f2=sin(x);       // Обчислюване значення функції в точці x
   pf=(f1-f2) /dx; // Знаходимо значення похідної
   std::cout << "dsin(x) /dx=" << pf <<" x= "<<x;
   getchar();
   return 0;
}
```

Результат роботи програми наведений на рис. 13.

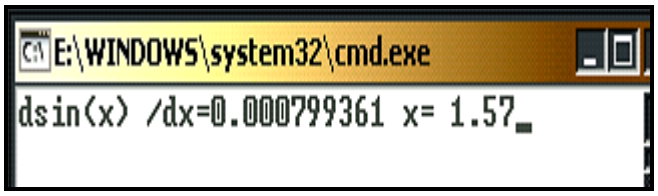


Рис. 13. **Результат роботи програми**

Другий варіант написання програми. Виведення результатів в стилі Си (за допомогою функції printf):

```
// Включення заголовних файлів
#include <math.h>
#include <conio.h>
#include <stdio.h>
int main()
{
   double f1,f2;  // Значення функцій
      double  pf;     // Значення похідної
      double x;       // Значення аргументу функції
```

```
    double dx;     // Значення приросту аргументу
  dx=1.0e-11;
    // Привласнення значення аргументу
  x=1.57;
    // Обчислення значення функцій у двох точках
  f1=sin(x+dx);
  f2=sin(x);
    // Обчислення похідної
  pf=(f1-f2) /dx;
    // Виведення похідної та аргументу на екран
  printf("dsin(x) /dx=%6.4f x=%4.2f",pf,x);
  getch();
  return 0;
}
```

Завдання 1 до ч. 2 лабораторної роботи № 1. Скласти програму для обчислення похідної у заданій точці згідно з отриманим варіантом (табл. 4).

Таблиця 4

Варіанти завдань № 1 до лабораторної роботи № 1

№	Завдання	№	Завдання
1	2	3	4
1	$f(x)=\frac{\sqrt{x+1}}{\sqrt{x+1}+1};\ f'(0)=?$	11	$f(x)=\frac{\cos x}{1+\sin x};\ f'(\pi/2)=?$
2	$f(x)=\sin 4x\cos 4x;\ f;(\pi/3)=?$	12	$f(x)=\sin^2 x^2;\ f'(0)=?$
3	$f(x)=\sin^4 x-\cos^4 x;\ f'(\pi/12)=?$	13	$f(x)=\sin^3\frac{x}{2};\ f'(\pi/2)=?$
4	$f(x)=\frac{\sqrt{x-1}+\sqrt[5]{x-1}}{\sqrt[3]{x-1}};\ f'(2)=?$	14	$f(x)=\sqrt{\frac{1-x}{1+x^2}};\ f'(0)=?$
5	$f(x)=5(x+1)^2\ \sqrt[5]{x-1};\ f'(2)=?$	15	$f(x)=\sqrt{x^2-1+\sqrt[3]{x}};\ f'(1)=?$

Закінчення табл. 4

1	2	3	4
6	$f(x)=\frac{1}{2}\sin x\ tg2x;\ f'(\pi/2)=?$	16	$f(x)=\frac{2^{2x}}{\sqrt{2-2^{2x}}};\ f'(0)=?$
7	$f(x)=2^{x-2x^2-1};\ f'(0)=?$	17	$f(x)=(x^2-x)\cos^2 x;\ f'(0)=?$
8	$f(x)=\frac{\sin 2x}{\sqrt{x}};\ f'(\pi)=?$	18	$f(x)=\frac{x^2+3}{x-1};\ f'(0)=?$
9	$f(x)=\frac{x}{\sqrt{x^2+3}}+\frac{1}{x+1};\ f'(1)=?$	19	$f(x)=\frac{x-2}{\sin^2 x};\ f'(\pi/2)=?$
10	$f(x)=\frac{x}{3}-\frac{3}{x};\ f'(3)=?$	20	$f(x)=x-\frac{2}{x^2}-\frac{1}{3x^3};\ f'(-1)=?$

Завдання 2. Скласти програму для знаходження значень виразів A і B по заданих значеннях початкових даних x, y, z.

Початкові дані (за варіантами) знаходяться в табл. 5.

Таблиця 5

Варіанти завдань № 2 до лабораторної роботи № 1

№ вар.	Функції	Початкові дані		
		x	***y***	***z***
1	2	3	4	5
1	$A=2^{-x}\sqrt{x+\sqrt[4]{\lvert y\rvert}},\ B=\sqrt[3]{e^{x-1/\sin z}}$	3,98	-1,62	0,52
2	$A=y^{\sqrt[3]{\lvert x\rvert}}+\sin^3(y-3),\ B=\frac{y(\mathrm{arctg}z-\pi/6)}{\lvert x\rvert+1/(y^2+1)}$	-0,62	0,82	25
3	$A=2^{(y^x)}+3^{(y^x)},\ B=\frac{\lvert x-y\rvert(1+\sin^2 z/(x+y))}{e^{\lvert x-y\rvert}+0.5x}$	3,25	0,32	0,4
4	$A=\frac{\sqrt{\lvert x-1\rvert}-\sqrt[3]{\lvert y\rvert}}{1+0.5x^2+0.25y^2},\ B=x\left(\mathrm{arctg}z+e^{-(x+3)}\right)$	-0,62	3,32	5,4

Продовження табл.5

1	2	3	4	5
5	$A=\sqrt{y+\sqrt[3]{x-1}},\ B=\lvert x-y\rvert\left(\sin z^2+\operatorname{tg}z\right)$	17,4	10,3	0,82
6	$A=\dfrac{y^{x+1}}{\sqrt[3]{\lvert y-2\rvert}+3}+\dfrac{x+0.5y}{2\lvert x+y\rvert},\ B=\left(x+1\right)^{-1/\sin z}$	1,62	-15,4	0,25
7	$A=\dfrac{x^{y+1}+e^{y-1}}{1+x\lvert y-\operatorname{tg}z\rvert},\ B=1+\lvert y+x\rvert+\dfrac{\lvert y-x\rvert^2}{2}-\dfrac{\lvert y-x\rvert^3}{3}$	2,44	0,86	-0,16
8	$A=1+x+\dfrac{x^2}{2!}+\dfrac{x^3}{3!}+\dfrac{x^4}{4!},\ B=x\left(\operatorname{arctg}z+\cos^3 y\right)$	0,33	0,02	32
9	$A=\left(1-x\right)\dfrac{x+y/\left(x^2+4\right)}{y^{x-2}+1/\left(x^2+4\right)},\ B=\dfrac{1+\cos\left(y-2\right)}{0.5x+\sin^2 z}$	3,25	4,2	-0,66
10	$A=y+\dfrac{x}{y+x^2/\left(y+x^3/y\right)},B=\left(1+\operatorname{tg}^2 z/2\right)^{\sqrt{\lvert y\rvert+6}}$	0,12	-8,75	0,76
11	$A=\lg\left(\sqrt{e^{x-y}}+x^{\lvert y\rvert}+z\right),\ B=x-\dfrac{x^3}{3!}-\dfrac{x^5}{5!}$	1,53	-3,26	8,2
12	$A=\dfrac{2\cos\left(x-\pi/6\right)}{0.5+\sin^2 y},\ B=1+\dfrac{z^2}{3+z^2/5}$	1,42	-1,22	3,52
13	$A=\dfrac{\sqrt[3]{8+\lvert x-y\rvert^2+1}}{x^2+y^2+2},\ B=\cos^2\left(\operatorname{arctg}0.5\right)$	3,74	-0,82	0,16
14	$A=\dfrac{1+\sin^2\left(x+y\right)}{\left\lvert x-2y/\left(1+x^2y^2\right)\right\rvert}x^{\lvert y\rvert},\ B=e^{\lvert x-y\rvert}+\left(\operatorname{tg}^2 z+1\right)^x$	3,74	-0,82	0,16
15	$A=\lvert\cos x+\cos y\rvert^{1+2\sin y},\ B=1+z+\dfrac{z^2}{2}+\dfrac{z^3}{3}$	0,42	-0,87	-0,47
16	$A=\ln\left(y-\sqrt{\lvert y\rvert}\right)\left(x-y/2\right),\ B=\sin^2\operatorname{arctg}z$	-15,2	4,64	21,3
17	$A=\sqrt{10\left(\sqrt[3]{x}+x^{y+2}\right)},\ B=\left(\arcsin z\right)^2+\lvert x+y\rvert$	16,5	-2,75	0,15
18	$A=5\operatorname{arctg}x-0.25\operatorname{arctg}y,\ B=\dfrac{x+3\lvert x-y\rvert+x^2}{\lvert x-y\rvert^z+z^2}$	-17,2	6,33	3,25

Закінчення табл.5

1	2	3	4	5
19	$A = e^{\lvert x+y\rvert} + \lvert x-y\rvert^{x+y},\ B = \operatorname{arctg} x + \operatorname{arctg} z$	-2,23	-0,82	15,2
20	$A = \lvert x^{y/x} - \sqrt{y/x}\rvert,\ B = (y-x)\dfrac{y - z/(y-x)}{1+(y+x)^2}$	1,82	18,5	-3,29

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 – придбання практичних навиків з підготовки, відладки 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 умовній операції ?:.

Після виконання лабораторної роботи студент повинен вміти розробляти і відладжувати 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 оператора вибору: if, if/else, switch/case і умовній 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 першої та другої форми реалізують алгоритми, представлені на рис. 14.

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

Рис.14. **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: операторN+1;
}
```

Оператор switch/case реалізує алгоритм, наведений на рис. 15.

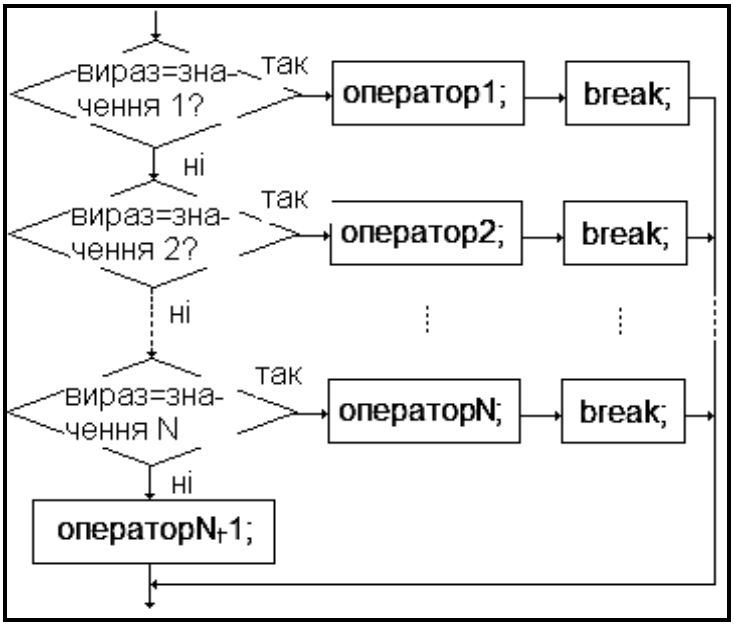


Рис. 15. **Алгоритми роботи оператора switch/case**

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.

Оператора 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 до лабораторної роботи № 2

Завдання 1. Знайти всі раціональні корені полінома n-го ступеня з цілими коефіцієнтами.

Змістовний сенс практичного завдання – обчислення дісних коренів многочленів.

При вирішенні таких завдань використовується теорема.

Теорема. Для того, щоб нескоротний дріб p/q був коренем рівняння ($q \neq 0$) $a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \cdots + a_0 = 0 \quad a_n \neq 0$ з цілими

коефіцієнтами, необхідно, щоб число p було дільником вільного члена a_0, а число q – дільником старшого коефіцієнта an.

Якщо рівняння має цілі коефіцієнти, а старший коефіцієнт дорівнює одиниці (тобто a_n=1), то раціональним корінням цього рівняння можуть бути тільки цілі числа, які є дільниками вільного члена a_0.

Приклад

$f(x) = 6x^4 – x^3 – 7x^2 + x + 1$.

Вільний член цього рівняння має два дільника ±1. Старший коефіцієнт рівняння має вісім дільників ±1, ±2, ±3, ±6. Складемо всі можливі дроби p/q. Отримаємо такі числа 1 -1, 1/2 -1/2, 1/3 -1/3, 1/6, -1/6.

Усі раціональні корені початкового рівняння належать цій безлічі чисел.

Текст програми:

```
#include <math.h>
#include <conio.h>
#include <stdio.h>
int main()
{
   double x,f;
      x=1.;
   f=6.*pow(x,4.)-pow(x,3.)-7.*pow(x,2.)+x+1.;
   if(f==0.)     printf("f(x) =%6.2f x=%6.2f\n",f,x);

      x=-1.;
   f=6.*pow(x,4.)-pow(x,3.)-7.*pow(x,2.)+x+1.;
   if(f==0.)     printf("f(x) =%6.2f x=%6.2f\n",f,x);

      x=1./6.;
   f=6.*pow(x,4.)-pow(x,3.)-7.*pow(x,2.)+x+1.;
   if(f==0.)     printf("f(x) =%6.2f x=%6.2f\n",f,x);

      x=-1./6.;
   f=6.*pow(x,4.)-pow(x,3.)-7.*pow(x,2.)+x+1.;
   if(f==0.)     printf("f(x) =%6.2f x=%6.2f\n",f,x);
```

```
    x=1./3.;
  f=6.*pow(x,4.)-pow(x,3.)-7.*pow(x,2.)+x+1.;
  if(f==0.)     printf("f(x) =%6.2f x=%6.2f\n",f,x);

    x=-1./3.;
  f=6.*pow(x,4.)-pow(x,3.)-7.*pow(x,2.)+x+1.;
  if(f==0.)     printf("f(x) =%6.2f x=%6.2f\n",f,x);

    x=1./2.;
  f=6.*pow(x,4.)-pow(x,3.)-7.*pow(x,2.)+x+1.;
  if(f==0.)     printf("f(x) =%6.2f x=%6.2f\n",f,x);

    x=-1./2.;
  f=6.*pow(x,4.)-pow(x,3.)-7.*pow(x,2.)+x+1.;
  if(f==0.)     printf("f(x) =%6.2f x=%6.2f\n",f,x);
  getch();
  return 0;
}
```

Результат роботи програми наведений на рис. 16.

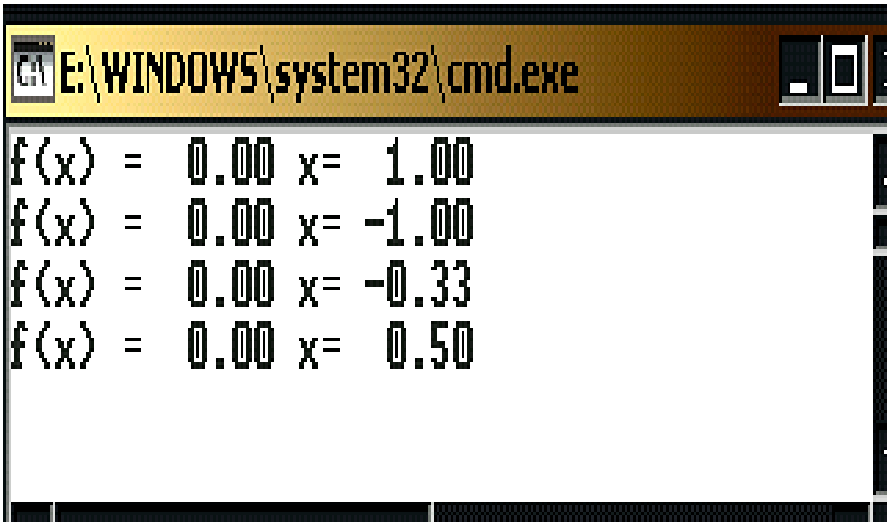


Рис. 16. **Результат роботи програми**

Варіанти до завдання № 1. Знайти корені рівнянь згідно з обраними варіантами (табл. 6).

Таблиця 6

Варіанти завдань № 1

№	Завдання
1	$f(x)=x^3-4x^2-x+4$
2	$f(x)=6x^4+7x^3-36x^2-7x+6$
3	$f(x)=2x^4+7x^3-12x^2-38x+21$
4	$f(x)=x^3+2x^2-x-2$
5	$f(x)=2x^3-4x^2-x-15$
6	$f(x)=6x^4+19x^3-7x^2-26x+12$
7	$f(x)=2x^5+5x^4+11x^3+14x^2+11x+5$
8	$f(x)=x^4+4x^3-2x^2-12x+9$
9	$f(x)=2x^3-3x^2-3x+2$
10	$f(x)=x^5+3x^4-x^3+2x^2-24x-32$
11	$f(x)=3x^3+2x^2-2x+3$
12	$f(x)=x^4+3x^3-44x^2+15x+25$
13	$f(x)=2x^3-4x^2-x-15$
14	$f(x)=6x^4+19x^3-7x^2-26x+12$
15	$f(x)=24x^5+10x^4-x^3-19x^2-5x+6$
16	$f(x)=12x^5+18x^4-45x^3-45x^2+18x+12$
17	$f(x)=2x^5+5x^4+11x^3+14x^2+11x+5$
18	$f(x)=x^4+2x^3-2x^2-6x+5$
19	$f(x)=6x^4+7x^3-36x^2-7x+6$
20	$f(x)=6x^4+5x^3-38x^2+5x+6$

Завдання 2. Обчислити значення функції Y(x) при різних значеннях початкових даних x і a.

Вид функції і значення початкових даних наведені в табл. 7.

Таблиця 7

Вид функції і значення початкових даних

№ вар.	Функція	Початкові дані	
		x	***a***
1	2	3	4
1	$Y = \frac{1+a^x}{1-a^x} + \begin{cases} 1+\mathrm{tg}(x/2), & x > a \\ 1+\mathrm{tg}(2x), & x \le a \end{cases}$	3,1 1,3	2,3
2	$Y = \frac{\sqrt{1+e^x}}{\sqrt{1-e^x}} + \begin{cases} \sqrt{a+x/2}, & x > a \\ 1/(a+x/2), & x \le a \end{cases}$	7,3 2,8	5,8
3	$Y = \frac{\sqrt{1+a^x}}{\sqrt{1-a^x}} + \begin{cases} \mathrm{tg}(a+x), & x > a/2 \\ \mathrm{tg}(a/x+1), & x \le a/2 \end{cases}$	4,2 1,25	3,7
4	$Y = \mathrm{tg}\frac{1-x}{1+x} + \begin{cases} a+e^{(1-x)}, & x \ge a \\ 1-e^{(1-x)}, & x < a \end{cases}$	0,5 0,34	0,45
5	$Y = \frac{25+\sqrt{\lvert x\rvert}}{25a} + \begin{cases} a/\sqrt{\lvert x\rvert}, & x < a \\ \sqrt{\lvert x\rvert + a}, & x \le a \end{cases}$	3,7 7	6,2
6	$Y = \frac{\sin 5x}{1+5x} + \begin{cases} a+\sin^2 x, & x > a \\ 1/(a+\sin^2 x), & x \le a \end{cases}$	2,5 0,17	π/20
7	$Y = \frac{ax}{1+\sqrt{\lvert x\rvert}} + \begin{cases} (x+1)a, & x > a \\ 1+(x+1)a/2x, & x \le a \end{cases}$	2,8 1,73	2,1
8	$Y = \frac{x^3}{1+e^{-3x}} + \begin{cases} a^{3x}/(11+x), & a/x \ge \pi/6 \\ a^{3x}/(11-x), & a/x < \pi/6 \end{cases}$	2,75 12,3	5
9	$Y = (a+1)^5 + \begin{cases} a+\sqrt{1-e^{-x}}, & x \ge a \\ a-\sqrt{1+e^x}, & x < a \end{cases}$	2,8 1,1	1,25

1	2	3	4
10	$Y = x^4 e^{(x+1)} + \begin{cases} ax/\left(1+\ln(x+1)\right), & x > a \\ a(x+1)/\left(1+\ln(x+1)\right), & x \le a \end{cases}$	2,75 1,89	2,3
11	$Y = \dfrac{\operatorname{tg} x}{1+x} + \begin{cases} a/\left(1+x^3\right), & x > \operatorname{tg}(a) \\ ax/\left(1+x^3\right), & x \le \operatorname{tg}(a) \end{cases}$	4,6 1,27	1,3
12	$Y = \dfrac{x}{1+x} + \begin{cases} a\sin(1+x), & x > a \\ a\cos(1+x), & x \le a \end{cases}$	5,85 2,13	4,3
13	$Y = \dfrac{\ln\left(1+x^2\right)}{1+x^2} + \begin{cases} 1+a^x, & x > a \\ 1-a^{-x}, & x \le a \end{cases}$	7,85 5,2	8,5
14	$Y = \dfrac{1+\sqrt{x}}{\left(1+\sqrt{x}\right)^2} + \begin{cases} 1+\sqrt{x^a}, & a \ge x \\ 1+\sqrt{x-a}, & a < x \end{cases}$	4,3 6,75	5,8
15	$Y = \dfrac{\ln\left(1+\sqrt{1+x^2}\right)}{1+e^{(1+x*x)}} + \begin{cases} a+e^{(1+x*x)}, & x > a \\ a-e^{(1+x*x)}, & x \le a \end{cases}$	6 2,7	4,5
16	$Y = \dfrac{4+5\operatorname{tg}(1+3x)/2}{5.5a} + \begin{cases} e^{2x-1}+a, & x \le a \\ e^{2x-1}-a, & x > a \end{cases}$	0,75 3,4	1,2
17	$Y = \dfrac{a\cos^2 x}{1+\operatorname{tg}^2(x/4)} + \begin{cases} \left(x^2+1\right)e^x, & e^x > a \\ \left(x^2-1\right)e^x, & e^x \le a \end{cases}$	1,7 0,6	3,7
18	$Y = \dfrac{(x+10)a}{x-10} + \begin{cases} (60-x)x/\left(60+x^2\right), & x > a \\ (x-60)x/\left(60+x^2\right), & x \le a \end{cases}$	10,2 5,4	8,7
19	$Y = \dfrac{1+x^2+x^3}{1+\sqrt{\lvert x\rvert}+\sqrt[3]{\lvert x\rvert}} + \begin{cases} \left(e^x-1\right)a, & x > a \\ \left(1+e^x\right)a, & x \le a \end{cases}$	7,3 4,75	5,1
20	$Y = \dfrac{(x+10)a}{x-10} + \begin{cases} (60-x)x/\left(60+x^2\right), & x > a \\ (x-60)x/\left(60+x^2\right), & x \le a \end{cases}$	1,2 8,4	5,7

Завдання 3 (на оцінку 12). У східному календарі прийнятий 60-річний цикл, що складається з 12-літніх підциклів, що позначаються назвами кольору: зелений, червоний, жовтий, білий і чорний. У кожному підциклі роки носять назви тварин: щура, корови, тигра, зайця, дракона, змії, коня, вівці, мавпи, курки, собаки і свині. За номером року вивести його назву, якщо 1984 рік був початком циклу – роком зеленого щура. Використувати оператор switch/case.

Контрольні питання

1. Що таке обчислювальний процес, що 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 – придбання практичних навиків з підготовки, відладки та виконання циклічних програм.

Перед виконанням лабораторної роботи студент повинен знати: основи додатка 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 декілька відрізняється від попереднього. Цикл може виконуватися задане заздалегідь число кроків, а може завершуватися при настанні деякої умови.

Існує три види циклів: while, for і do.

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

while (вираз) тіло циклу;

Оператора while реалізує алгоритм, представлений на рис. 17.

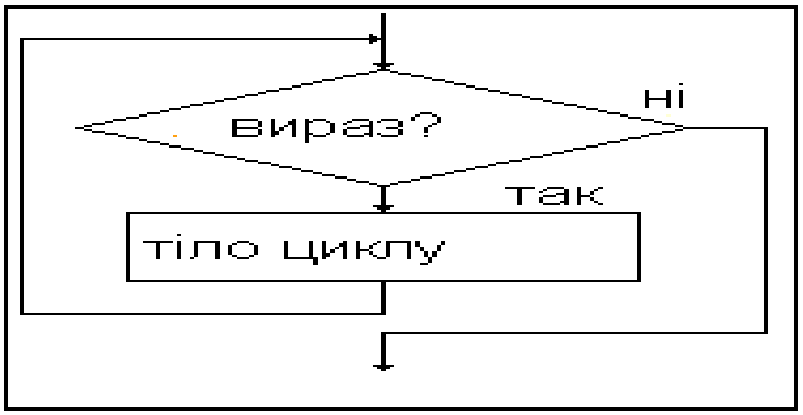


Рис. 17. **Алгоритми роботи оператора while**

Як вираз, допускається використовувати будь-який вираз мови C++, 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.

Тіло циклу виконується до тих пір, поки значення виразу рівне true. 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. 18.

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 відмінно 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.

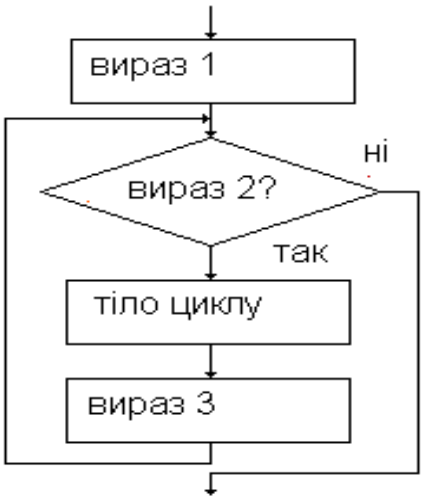


Рис. 18. **Алгоритми роботи оператора for**

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 помилковою.

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:

for (;вираз 2;) тіло циклу;

з опущеними вираз 1 і вираз 3 еквівалентний циклу

while (вираз 2) тіло циклу;

Цикл for:

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 умовою поста 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 помилковою, можна використовувати оператора break.

Оператор do while реалізує алгоритм, наведений на рис. 19.

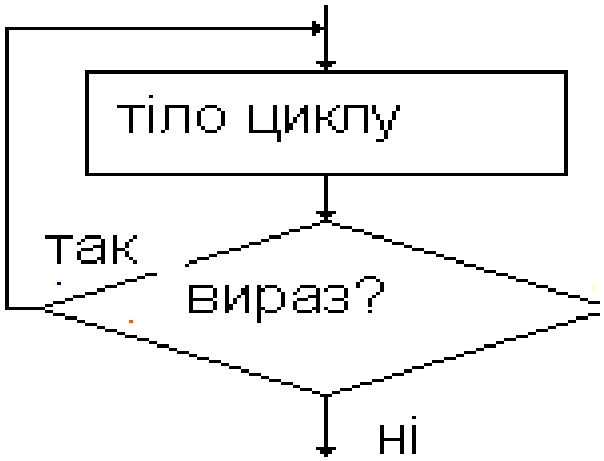


Рис. 19. **Алгоритм роботи оператора do while**

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 1

Обчислити площу фігури, обмеженої лініями (рис. 20).

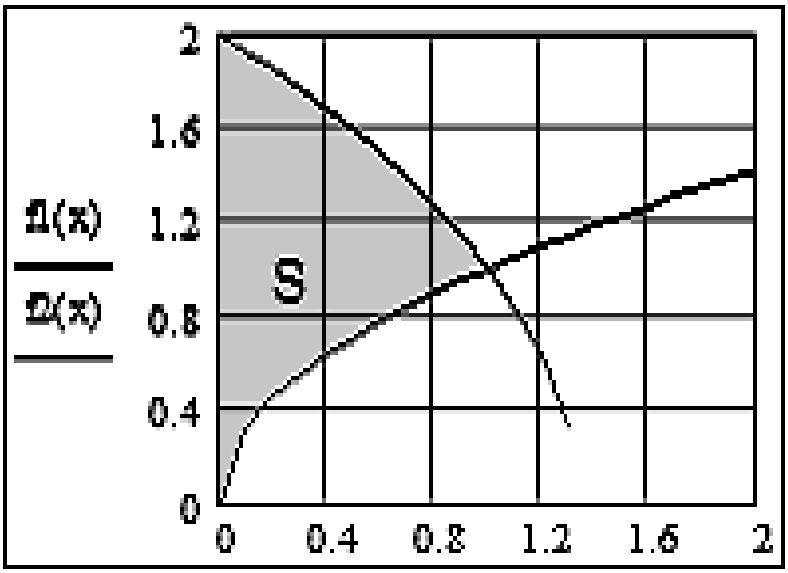


Рис. 20. **Площа фігури обмежена лініями**

Для обчислення площі фігури обмеженою лініями можна використовувати ітераційний вираз $s_{i+1} = s_i + (f2(x_i) - f1(x_i))\ dx$, де $s_{i+1} = s_i + (f2(x_i) - f1(x_i))\ dx$, $s_{i+1} = s_i + (f2(x_i) - f1(x_i))\ dx$ ¡ $i = 0, 1, 2 .. n$.

Текст програми:

```
#include <math.h>
#include <stdio.h>
#include <windows.h>
int main()
{
  char buf[80];
  double x,dx,f1,f2,s;
  s=0.;
  x=0.;
dx=1.0e-5;
f1=sqrt(x);
f2=sqrt(4.-3.*x);
for(;f1<f2;x=x+dx)
 {
   f1=sqrt(x);
   f2=sqrt(4.-3.*x);
```

```
   s=s+(f2-f1)*dx;
 }
  sprintf(buf," s=%6.2f\n x=%6.2f\n f1=%6.2f\n f2=%6.2f\n", s, x, f1, f2);
  printf("%s",buf);
  return 0;
}
```

Результат роботи програми наведений на рис. 21.

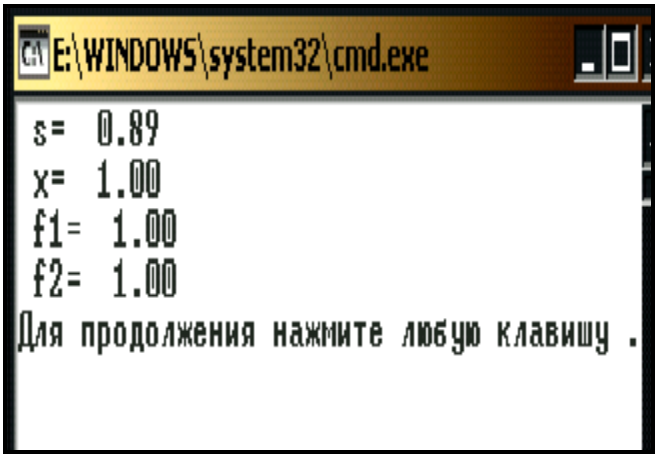
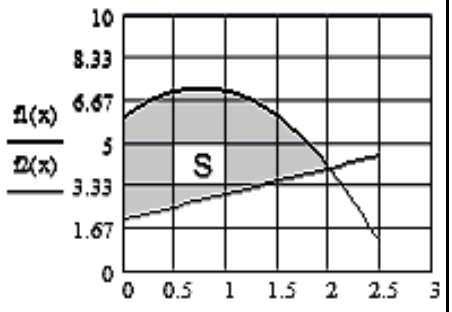
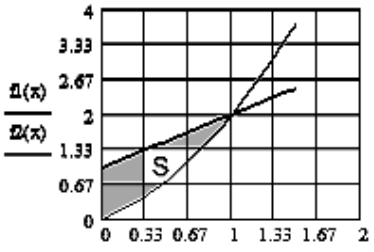


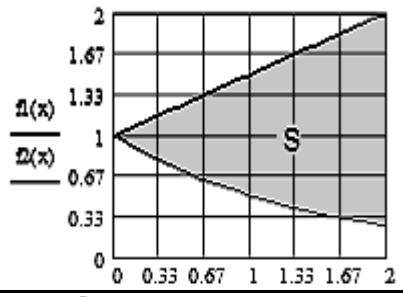
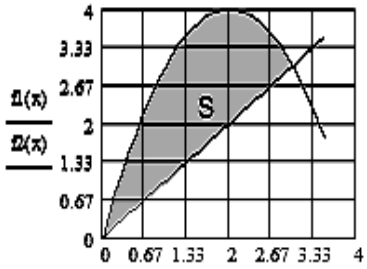
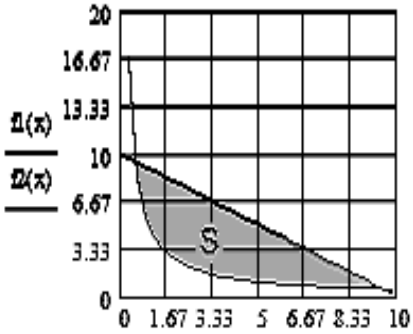
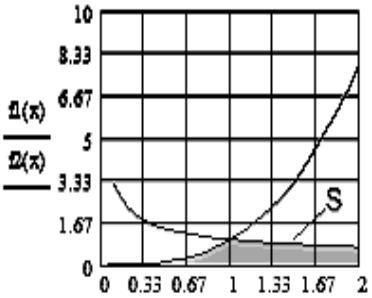
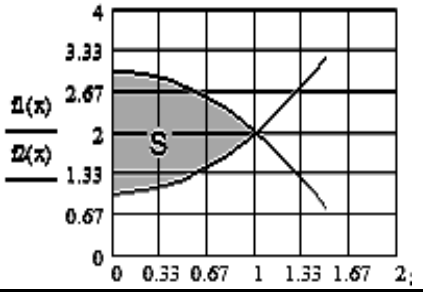
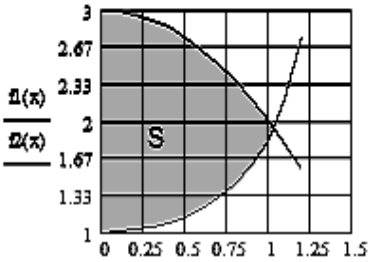
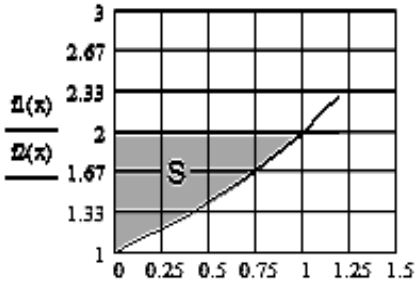
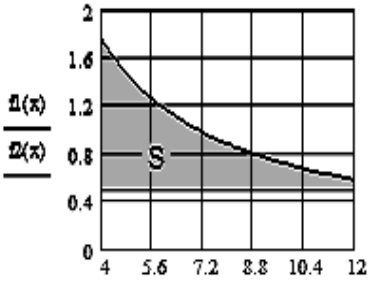
Рис. 21. **Результат роботи програми**

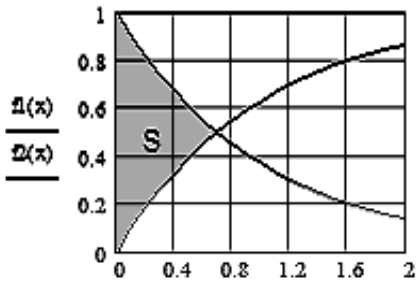
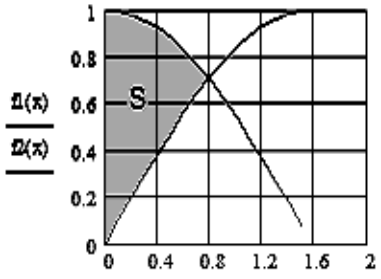
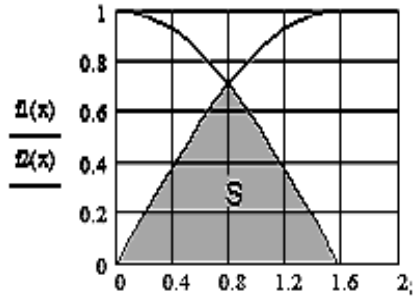
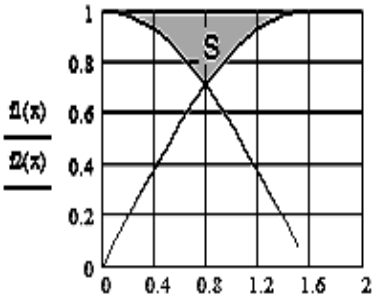
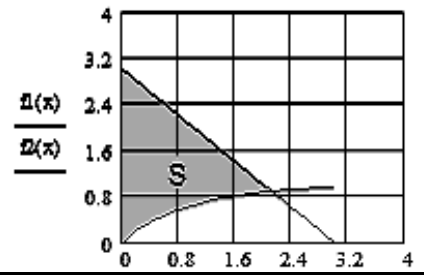
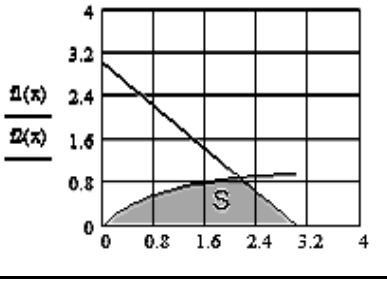
Варіанти до завдання № 1. Знайти площу фігури, що обмежена заданими лініями згідно з варіантами (табл. 8.).

Таблиця 8

Варіанти завдань № 1

№	Завдання	Цикл	№	Завдання	Цикл
1	2	3	4	5	6
1	$f1(x) = -2 \cdot x^2 + 3 \cdot x + 6$ $f2(x) = x + 2$ 	for	9	$f1(x) = x^2 + x$ $f2(x) = x + 1$ 	do while

1	2	3	4	5	6
2	$f1(x) = \left(\frac{1}{2}\right)^x$ $f2(x) = 1 + \frac{x}{2}$ 	do while	10	$f1(x) = 4 \cdot x - x^2$ $f2(x) = x$ 	for
3	$f1(x) = \frac{5}{x}$ $f2(x) = 10 - x$ 	for	11	$f1(x) = x^3$ $f2(x) = \frac{1}{x^{0.5}}$ 	for
4	$f1(x) = x^2 + 1$ $f2(x) = 3 - x^2$ 	while	12	$f1(x) = \frac{1}{\cos(x)}$ $f2(x) = 3 - x^2$ 	while
5	$f1(x) = 2^x$ $f2(x) = 2$ 	do while	13	$f1(x) = \frac{7}{x}$ $f2(x) = 0.5$ 	do while

1	2	3	4	5	6
6	$f1(x) = e^{-x}$ $f2(x) = 1 - e^{-x}$ 	do while	14	$f1(x) = \sin(x)$ $f2(x) = \cos(x)$ 	do while
7	$f1(x) = \sin(x)$ $f2(x) = \cos(x)$ 	for	15	$f1(x) = \sin(x)$ $f2(x) = \cos(x)$ 	for
8	$f1(x) = 1 - e^{-x}$ $f2(x) = -x + 3$ 	while	16	$f1(x) = 1 - e^{-x}$ $f2(x) = -x + 3$ 	while

Завдання 2. Обчислення функції за допомогою розкладання в ряд.

Обчислити й вивести на екран у вигляді таблиці значення функції, заданої за допомогою ряду Тейлора, на інтервалі від $x_{нач}$ до $x_{кон}$ з кроком dx з точністю ε. Таблицю забезпечити заголовком і шапкою. Кожен рядок таблиці повинен містити значення аргументу, значення функції та кількість підсумованих членів ряду.

Варіант 1

$$\ln\left(\frac{x+1}{x-1}\right) = 2\cdot\sum_{n=0}^{\infty}\frac{1}{(2\cdot n+1)\cdot x^{2\cdot n+1}} = 2\cdot\left(\frac{1}{x}+\frac{1}{3\cdot x^3}+\frac{1}{5\cdot x^5}+\ldots..\right) \quad |x| > 1.$$

Варіант 2

$$e^{-x} = \sum_{n=0}^{\infty}\frac{(-1)^n\cdot x^n}{n!} = 1-x+\frac{x^2}{2!}-\frac{x^3}{3!}+\frac{x^4}{4!}-\ldots.. \qquad |x| < \infty.$$

Варіант 3

$$\ln(x) = 2\cdot\sum_{n=0}^{\infty}\frac{(x-1)^{2\cdot n+1}}{(2\cdot n+1)\cdot(x+1)^{2\cdot n+1}} = 2\cdot\left[\frac{x-1}{x+1}+\frac{(x-1)^3}{3\cdot(x+1)^3}+\frac{(x-1)^5}{5\cdot(x+1)^5}+\ldots.\right].$$

$$x > 0$$

Варіант 4

$$\ln(x+1) = \sum_{n=0}^{\infty}\frac{(-1)^n\cdot x^{n+1}}{n+1} = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots.. \qquad -1 < x \le 1.$$

Варіант 5

$$\ln\left(\frac{1+x}{1-x}\right) = 2\cdot\sum_{n=0}^{\infty}\frac{x^{2\cdot n+1}}{2\cdot n+1} = 2\cdot\left(x+\frac{x^3}{3}+\frac{x^5}{5}+\ldots..\right) \qquad |x| < 1.$$

Варіант 6

$$\ln(1-x) = -\sum_{n=1}^{\infty}\frac{x^n}{n} = -\left(x+\frac{x^2}{2}+\frac{x^4}{4}+\ldots..\right) \quad -1 \le x < 1.$$

Варіант 7

$$\operatorname{acot}(x) = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1} \cdot x^{2 \cdot n+1}}{2 \cdot n + 1} = \frac{\pi}{2} - x + \frac{x^3}{3} - \frac{x^5}{5} + \ldots\ldots \qquad |x| \le 1.$$

Варіант 8

$$\operatorname{atan}(x) = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2 \cdot n + 1) \cdot x^{2 \cdot n+1}} = \frac{\pi}{2} - \frac{1}{x} + \frac{1}{3 \cdot x^3} - \frac{1}{5 \cdot x^5} + \ldots.$$

$$x > 1.$$

Варіант 9

$$\operatorname{atan}(x) = \sum_{n=0}^{\infty} \frac{(-1) \cdot x^{2 \cdot n+1}}{2 \cdot n + 1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots\ldots \qquad |x| \le 1.$$

Варіант 10

$$\operatorname{atanh}(x) = \sum_{n=0}^{\infty} \frac{x^{2 \cdot n+1}}{2 \cdot n + 1} = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \ldots. \qquad |x| < 1.$$

Варіант 11

$$\operatorname{atanh}(x) = \sum_{n=0}^{\infty} \frac{1}{(2 \cdot n + 1) \cdot x^{2 \cdot n+1}} = \frac{1}{x} + \frac{1}{3 \cdot x^3} + \frac{1}{5 \cdot x^5} + \ldots\ldots \qquad |x| > 1.$$

Варіант 12

$$\operatorname{acot}(x) = \frac{-\pi}{2} + \sum_{n=9}^{\infty} \frac{(-1)^{n+1}}{(2 \cdot n + 1) \cdot x^{2 \cdot n+1}} = \frac{-\pi}{2} - \frac{1}{x} + \frac{1}{3 \cdot x^3} - \frac{1}{5 \cdot x^5} + \ldots.$$

$$x < -1.$$

Варіант 13

$$e^{-x^2} = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2\cdot n}}{n!} = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} + \frac{x^8}{4!} - \ldots.$$

$$|x| < \infty.$$

Варіант 14

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2\cdot n}}{(2\cdot n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots.$$

$$|x| < \infty.$$

Варіант 15

$$\frac{\sin(x)}{x} = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2\cdot n}}{(2\cdot n+1)!} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + \ldots.$$

$$|x| < \infty.$$

Варіант 16

$$\ln(x) = 2\cdot \sum_{n=0}^{\infty} \frac{(x-1)^{2\cdot n+1}}{(2\cdot n+1)\cdot(x+1)^{2\cdot n+1}} = 2\cdot\left[\frac{x-1}{x+1} + \frac{(x-1)^3}{3\cdot(x+1)^3} + \frac{(x-1)^5}{5\cdot(x+1)^5} + ..\right]$$

$$x > 0.$$

Варіант 17

$$\ln(x) = \sum_{n=0}^{\infty} \frac{(-1)^n \cdot (x-1)^{n+1}}{n+1} = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \ldots.$$

$$0 < x \le 2.$$

Варіант 18

$$\ln(x) = \sum_{n=0}^{\infty} \frac{(x-1)^{n+1}}{(n+1)\cdot(x+1)^{n+1}} = \frac{x-1}{x} + \frac{(x-1)^2}{2\cdot x^2} + \frac{(x-1)^3}{3\cdot x^3} + \ldots..$$

$$x > \frac{1}{2}.$$

Варіант 19

$$\text{asin}(x) = x + \sum_{n=1}^{\infty} 1\cdot 3\cdot \ldots\ldots \cdot \frac{2\cdot n-1}{2\cdot 4\cdot \ldots\ldots 2\cdot n\cdot (2\cdot n)} \cdot x^{2\cdot n+1} = x +$$

$$+ \frac{x^3}{2\cdot 3} + \frac{1\cdot 3\cdot x^5}{2\cdot 4\cdot 5} + \frac{1\cdot 3\cdot 5\cdot x^7}{2\cdot 4\cdot 6\cdot 7} + \frac{1\cdot 3\cdot 5\cdot 7\cdot x^9}{2\cdot 4\cdot 6\cdot 8\cdot 9} + \ldots\ldots$$

$$|x| < 1.$$

Варіант 20

$$\text{acos}(x) = \frac{\pi}{2} - \left[x + \sum_{n=1}^{\infty} \frac{1}{2\cdot 4\cdot \ldots\ldots \cdot 2\cdot n\cdot (2\cdot n+1)} \cdot 3\cdot \ldots\ldots \cdot (2\cdot n-1)\cdot x^{2\cdot n+1} \right] =$$

$$= \frac{\pi}{2} - \left(x + \frac{x^3}{2\cdot 3} + \frac{1\cdot 3\cdot x}{2\cdot 4\cdot 5} + \frac{1\cdot 3\cdot 5\cdot x^7}{2\cdot 4\cdot 6\cdot 8} + \frac{1\cdot 3\cdot 5\cdot 7\cdot x^9}{2\cdot 4\cdot 6\cdot 8\cdot 9} + \ldots. \right)$$

$$|x| < 1.$$

Завдання 3 (на оцінку 12). Скласти програму обчислення коренів рівняння f(x)=0 з точністю EPS = 0,0001.

Інтервал локалізації кореня [a, b] відомий.

Використовувати:

метод простої ітерації;

метод половинного ділення відрізку [a, b] локалізації кореня;

метод Ньютона.

При складанні програми використати оператори циклу while і do while.

Примітка:

1. **Метод простої ітерації** полягає в тому, що за і-м наближенням кореня знаходиться і+1 наближення за формулою:

$$x_{i+1} = f(x_i), \quad i = 0, 1, 2, \ldots.$$

Процес продовжується до тих пір, поки відносна помилка для двох послідовних наближень не стане менша EPS:

$$\left| \frac{x_{i+1} - x_i}{x_i} \right| < EPS.$$

Процес ітерації сходиться на [a, b], якщо $|f'(x)| < 1$ при всіх $x \in (a,b)$.

Для початкового значення кореня x_0 прийняти $x_0 = (a,b)/2$.

2. **Метод половинного ділення.**

Завжди локалізований корінь знаходиться в тому інтервалі, на кінцях якого значення функції мають протилежні знаки. Якщо потрібно визначити корінь з точністю EPS, то ділення інтервалу навпіл продовжують, поки його довжина не стане менша 2*EPS. У цьому випадку середина останнього інтервалу дає значення кореня з потрібною точністю.

3. **Метод Ньютона** полягає у послідовному обчисленні за формулою:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \quad i = 0, 1, 2, \ldots.$$

Оцінка помилки к-го наближення кореня виконується так:

$$|f(x_k)/f'(x_k)| < EPS.$$

Початкове наближення кореня можна взяти дорівнюючим $x_0 = (a,b)/2$.

Знайти корінь рівняння за допомогою всіх трьох методів і порівняти отримані результати, для чого підставити набуте значення в рівняння. Обчислити кількість ітерацій, необхідних для знаходження кореня для кожного методу.

1. $f(x) = x - \sin x - 0{,}25 = 0$ (0; 2).
2. $x - 2{,}89\sin\frac{x}{3} - 0{,}126 = 0$ (1; 2).
3. $5x - 8\ln x - 8 = 0$ (3; 4).

4. $0{,}03125x^5 - 0{,}75x^2 - x + 4{,}005 = 0$ (1,5; 2,7).

5. $e^{-x} - x = 0$ (0,1; 0,6).

6. $x^3 + x - 1 = 0$ (0,5; 0,9).

7. $\sin x + x - 1 = 0$ (0; 1).

8. $\operatorname{tg} x - e^{-x} + x - 1 = 0$ (0,5; 1).

9. $0{,}5 \ln \frac{1+x}{1-x} + x - 1 = 0$ (0,1; 0,8).

10. $\cos 2x + 0{,}165x^3 - x = 0$ (0,1; 0,8).

11. $\operatorname{tg} \frac{x+1}{2} + x - 2 = 0$ (0,2; 1).

12. $0{,}5e^x + 0{,}5e^{-x} + x - 5 = 0$ (1,5; 2).

13. $\operatorname{arctg} x - x^2 = 0$ (0,5; 1).

14. $\operatorname{arctg} x + x - 1 = 0$ (0; 1).

15. $e^{-x^2} - x = 0$ (0,5; 1).

16. $x^6 + 0{,}008x^3 + 720x^2 + 720x - 720 = 0$ (1; 2).

17. $e^{-x} - x^2 + x = 0$ (0,5; 1,5).

18. $\ln\left(x + \sqrt{x^2 + 1}\right) + x - 2 = 0$ (0; 2).

19. $e^{-x} - x^2 + x = 0$ (1; 2).

20. $0{,}5e^x - 0{,}5e^{-x} + x - 2{,}3 = 0$ (0; 2).

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:

- структури 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 повертаного значення і список параметрів.

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

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

Розглянемо складові частини визначення.

За допомогою необов'язкового модифікатора клас можна явно задавати зона видимості функції, використовуючи ключові слова extern і static:

extern – глобальна видимість у всіх модулях програми (за замовчуванням);

static – видимість тільки в межах модуля, в якому визначена функція.

Тип повертаного функцією значення 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, передаваного у функцію, вказується його тип і ім'я (у 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 в 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 повертаного значення і типи параметрів спільно визначають тип функції.

Для виклику функції в простому випадку потрібно вказати її ім'я, за яким в круглих дужках через кому перераховуються імена передаваних аргументів. Виклик функції може знаходиться в будь-якому місці програми, де по синтаксису допустимий вираз того типу, який формує функція. Якщо тип повертаного значення функцією не void, то вона може

входити до складу виразу або, в окремому випадку, розташовуватися в правій частині оператора привласнення.

Нижче приведений приклад програми, що дозволяє знаходити максимальне число з трьох введених чисел, при цьому виклик функції здійснюється за її іменем (max3).

```
// Програма 4.1
// Приклад використання функції користувача max3 для пошуку
// максимального з трьох чисел
// Місце визначення функції – функція визначається
// безпосередньо  у  програмі
// (проте, можливий інший варіант визначення –  у заголовному файлі)
// Виклик функції – за іменем
// Повертане значення – використовується
// Метод передачі параметрів у функцію – за значенням
#include <iostream>
#include <conio.h>
#include <locale.h>

using namespace std;
double max3 (double, double, double);    // Оголошення прототипу функції

int main ()
{
system("CLS");
setlocale(0,"RUS");
double x, y, z;
cout << "\n ПОСЛІДОВНО ВВЕДІТЬ ТРИ ЧИСЛА: x,y,z: " << endl;
cin >> x;  cin >> y;  cin  >> z;
cout << "\n ВИ ВВЕЛИ: x = " << x << "\t y = " << y << "\t z = " << z ;
cout << "\n ТОМУ  max (x,y,z) = " << max3(x,y,z) << endl;
//                          |        |-> фактичні параметри
//                             |-> ім'я функції
return 0;
}
// Визначення функції max3
```

```
double   max3 (double a, double b, double c)
//             ---------------------------------------
//             |-> формальні параметри
 {
        double max = a;
        if ( b > max )
            max = b;
        if (c > max )
    max = c;
return max;
}
```

Один з можливих результатів виконання програми може мати такий вигляд (рис. 22.)

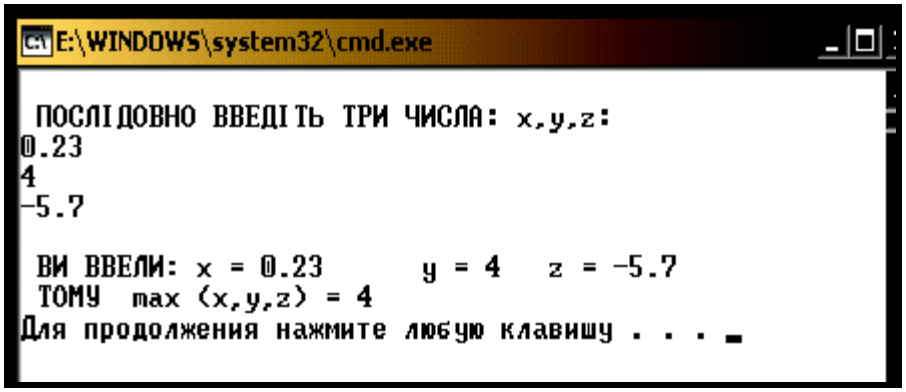


Рис. 22. **Результати пошуку максимального з 3 чисел**

Особливості виконання функції.

Усі величини, описані усередині 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, передуючий викличу функції, і адресу повернення з функції для того, щоб при виході з неї можна було продовжити виконання зухвалої функції.

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. Наступна програма ілюструє використання статичної локальної змінної в тілі функції.

```
// Програма 4.2
// Використання статичної локальної змінної в тілі функції
#include <iostream>
using namespace std;
void f (int a)  // визначення функції, що формує рядки з трьох
                // колонок з іменами   n, m, p
 {
   int m=0;
   cout << "\nn m p\n\n";  // виведення найменувань колонок
   while (a--)                  // формування рядків
     {
       static int n=0;
       int p=0;
       cout << n++ << ' ' << m++ << ' ' << p++ << '\n';
      }
 }
int main()
{ f(3);
  f(2);
  return 0;
}
```

Статична змінна n розміщується в сегменті даних і ініціалізується один раз при першому виконанні оператора, що містить її визначення. Автоматична змінна m ініціалізувалася при кожному вході у функцію. Автоматична змінна p ініціалізувалася при кожному вході в блок циклу.

Програма виведе на екран (рис. 23):

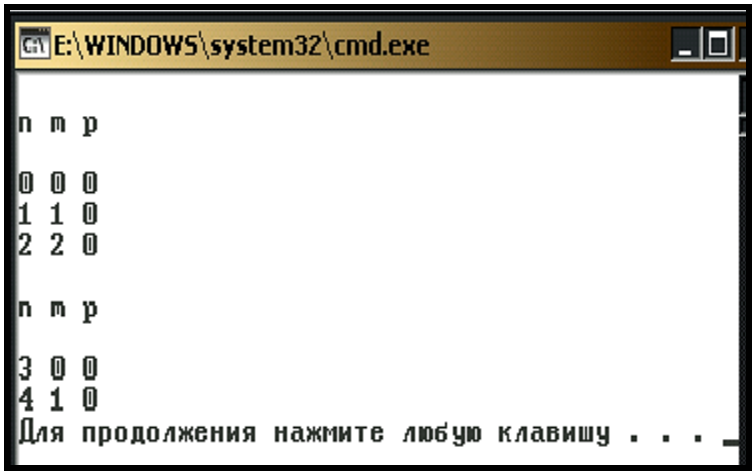


Рис. 23. **Використання статичної локальної змінної в тілі функції**

Повертане функцією значення.

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.

Оператора 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). Приклад використання декількох операторів return приведений в програмі 4.3.

```
// Програма 4.3
// Використання декількох операторів return в тілі функції
#include <locale.h>
#include <iostream>
using namespace std;

int Doubler(int); //об'ява функції, що подвоює значення аргументу
int main()
{
 int result = 0;
 int input;
 setlocale (0,"RUS");
 cout << "Введіть число між  0 і 10 : ";
 cin >> input;
 cout << "\n Перед викликом функції Doubler... ";
 cout << "\n Ваше число: " << input << " Результат: " << result << "\n";
 result = Doubler (input);
 cout << "\n Після виклику функції Doubler... ";
 cout << "\n Ваше число: " << input << " Результат: " << result << "\n";
```

```
 return 0;
}
int Doubler ( int x )   // Визначення функції Doubler
{
 if ( x <= 10 )  return  x * 2;
          else   return -1;
}
```

Результат роботи програми (рис. 24).

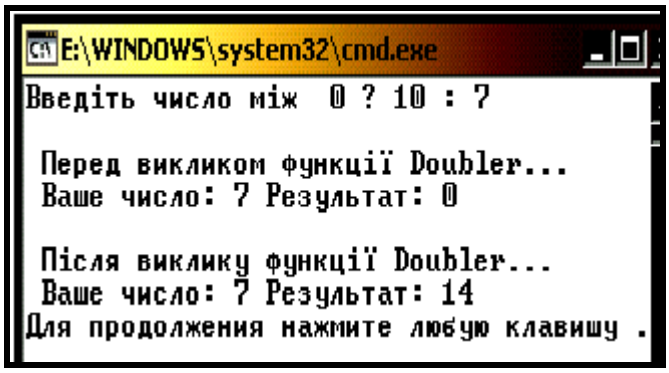


Рис. 24. **Реультат роботи програми 4.3**

Обмін інформацією між функціями.

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

- за допомогою глобальних змінних
- через параметри функції
- через повертані функцією значення.

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 відладку програми і перешкоджає приміщенню функцій в 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 операторові виклику функції – фактичними.

При виклику функції насамперед обчислюються вирази, що стоять на місці фактичних параметрів; потім в 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 по цих адресах і може змінити початкові значення параметрів.

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

```
// Програма 4.4
// 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 (рис. 25).

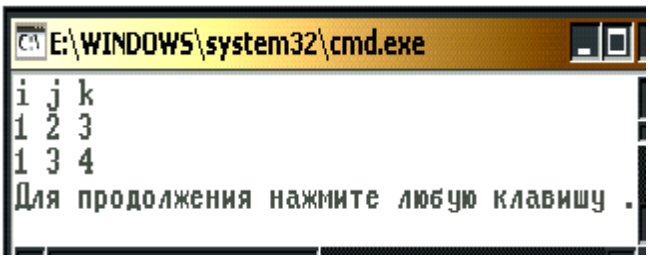


Рис. 25. **Реультат роботи програми 4.4**

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 передбачено. Це полегшує відладку великих програм, оскільки по заголовку функції можна зробити вивід про те, які величини в ній змінюються, а які немає. 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  myFunction (int, int);
int  myFunction (long, long);
int  myFunction (long);
```

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

Типи повертаних значень перевантажених функцій 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 повертаних значень, згенерує помилка компіляції.

Перевантаження функцій також називається поліморфізмом функцій. Поли (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 передаване 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 в програмі потрібної функції, що відповідає типу передаваних змінних. При виклику перевантаженої функції компілятор автоматично визначить, який саме варіант функції слід використовувати. Приклад використання перевантажених функцій показаний нижче.

```
// Програма 4.5
// Приклад поліморфізму функцій
// Чотири однойменні функції подвоюють значення аргументу
//кожного  з чотирьох типІв
#include <locale.h>
#include <iostream>
using namespace std;
int    Double (int);        // Прототипи однойменних функцій
long  Double (long);
float  Double (float);
double Double (double);

int main()
{
 int myInt        = 6500;    // Початкові дані різних типів
 long myLong  = 6500;
 float myFloat  = 6.5F;
```

```
 double myDouble = 6.5e20;

 int doubledInt;             // Оголошення змінних для результату
 long doubledLong;
 float doubledFloat;
 double doubledDouble;
 setlocale(0,"RUS");

 cout << "ПОЧАТКОВІ ДАННІ:\n";
 cout << "myInt: "    << myInt    << "\n";     // Виведення початкових даних
 cout << "myLong: "   << myLong   << "\n";
 cout << "myFloat: "  << myFloat  << "\n";
 cout << "myDouble: " << myDouble <<"\n\n";

 cout << "ФУНКЦІЇ, ЩО ВИКЛИКАЮТЬСЯ:\n";
 doubledInt    = Double (myInt);     // Виклики однойменних функцій
 doubledLong   = Double (myLong);
 doubledFloat  = Double (myFloat);
 doubledDouble = Double (myDouble);

  cout << "ВИВЕДЕННЯ РЕЗУЛЬТАТІВ:\n";
 cout << "doubledInt: " << doubledInt   << "\n";      // Виведення результатів
 cout << "doubledLong: "   << doubledLong   << "\n";
 cout << "doubledFloat: "  << doubledFloat  << "\n";
 cout << "doubledDouble: " << doubledDouble << "\n";

 return 0;
}

int Double (int original)
{
  cout << "ФУНКЦІЯ Double(int)\n";
  return 2 * original;
}

long Double (long original)
{
```

```
  cout << "ФУНКЦІЯ Double(long)\n";
  return 2 * original;
 }
float Double (float original)
{
  cout << "ФУНКЦІЯ Double(float)\n";
  return 2 * original;
}
double Double (double original)
{
  cout << "ФУНКЦІЯ Double(double)\n\n";
  return 2 * original;
}
```

Результат роботи програми (рис. 26).

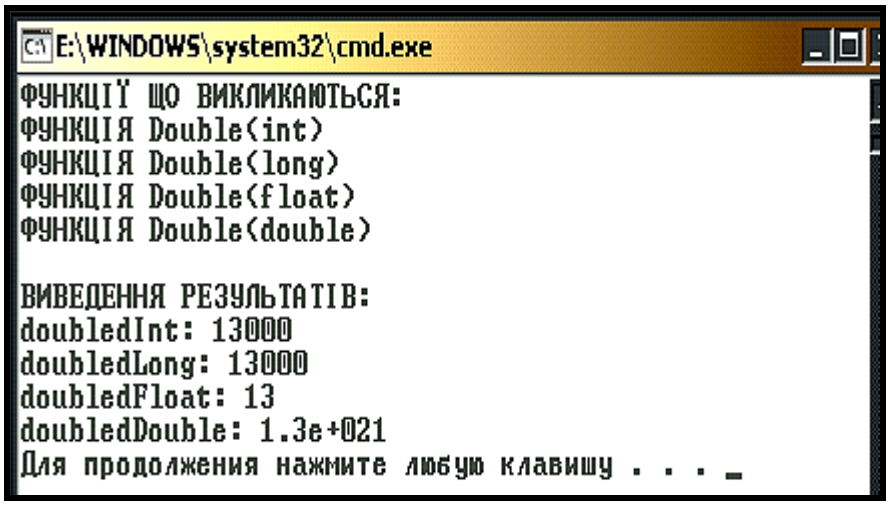


Рис. 26. **Результати програми з перевантаження функцій**

Функція Double() перевантажується для прийому чотирьох типів: int, long, float і double. По вигляду виклику функції Double() нічим не відрізняються один від одного. Проте, компілятор визначає тип переданого аргументу, на підставі якого вибирає відповідний варіант функції. Результат роботи цієї програми підтверджує очікувану черговість виклику варіантів цієї перевантаженої функції.

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 myFunction(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 myFunction(int x = 50);

Прототип такого оголошення потрібно розуміти таким чином. Функція myFunction(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 myFunction (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 myFunction (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 rnyFunction (int Paraml, int Param2, int Param3);

тоді параметру Param2 можна призначити стандартне значення тільки в тому випадку, якщо призначено стандартне значення і параметру Paramз. Параметру Paraml можна призначити стандартне значення тільки в тому випадку, якщо призначені стандартні значення як параметру Param2, так і параметру Paramз, Приклад використання значень, що задаються параметрам функцій за замовчуванням, показаний в програмі 4.6.

```
// Програма 4.6
// Використання стандартних значень параметрів (за замовчуванням)
#include <locale.h>
#include <iostream>
using namespace std;

int VolumeCube(int length, int width = 25, int height = 1);

int main()
{
 int length = 100;
 int width  = 50;
 int height = 2;
 int volume;

 setlocale(0,"RUS");
 volume = VolumeCube ( length, width, height );
 cout << "Перший об'єм рівний: " << volume << "\n";
 volume = VolumeCube ( length, width );
 cout << "Другий об'єм рівний: " << volume << "\n";
 volume = VolumeCube ( length );
 cout << "Третій об'єм рівний: " << volume << "\n";
 return 0;
}

int VolumeCube (int length, int width, int height )
 {
 return (length * width * height);
 }
```

Результат роботи програми (рис. 27).

Рис. 27. **Приклад використання стандартних значень параметрів**

У прототипі функції VolumeCube() оголошується, що функція приймає три параметри, причому останні два мають значення, що встановлюються за замовчуванням. Ця функція обчислює об'єм паралелепіпеда на підставі переданих розмірів. Якщо значення ширини не передане, то ширина встановлюється рівною 25, а висота – 1. Якщо значення ширини передане, а значення висоти немає, то за замовчуванням встановлюється тільки значення висоти. Але не можна передати у функцію значення висоти без передачі значення ширини.

У подальших рядках ініціалізувалися змінні, призначені для зберігання розмірів паралелепіпеда по довжині, ширині і висоті. Ці значення передаються функції VolumeCube(). Після обчислення об'єму паралелепіпеда результат виводиться у відповідному рядку. Далі функція VolumeCube() викликається знову, але без передачі значення для висоти. У цьому випадку для обчислення об'єму паралелепіпеда використовується значення висоти, задане за замовчуванням, і отриманий результат виводиться в іншому рядку.

При третьому виклику функції VolumeCube() не передається ні значення ширини, ні значення висоти. Тому замість них використовуються значення, задані за замовчуванням, і отриманий результат також виводиться на екран.

Створення власних заголовних файлів

Програміст може сам створювати потрібні йому заголовні файли з визначенням власних функцій.

Заголовні файли містять оголошення і визначення, загальні для різних програмних файлів, і тому часто створюються і включаються у файли програм директивою компілятора #include. Як такі оголошення і

визначення виступають класи, структури, об'єднання, типи, що перераховують, і прототипи функцій або їх визначення.

Директива #include використовується для включення копії вказаного в директиві файлу в те місці, де знаходиться ця директива. Найчастіше вона застосовується для включення в текст копії заголовного файлу.

Існують три форми директиви:

#include <ім'я_заголовочного_файла>

#include "ім'я_заголовочного_файла"

#include идентификатор_макроса

Відмінність між першими двома формами директиви полягає в методі пошуку препроцесором файлу, що включається.

1. Якщо ім'я файлу поміщене в кутові дужки, то послідовність пошуку препроцесором заданого файлу в каталогах визначається заданими каталогами включення (include directories).

2. Якщо ж ім'я файлу поміщене в лапки, препроцесор шукає файл, проглядаючи каталоги в такій послідовності:

- каталог того файлу, який містить директиву #include;
- поточний каталог;
- каталоги, вказані опцією компілятора.

Якщо ім'я файлу вказане з шляхом, то препроцесор ніде більше цей файл не шукає.

3. Третя форма директиви припускає наявність макросу, що визначає файл, що включається (у даній роботі ця форма не розглядається).

Як приклад, що ілюструє другу форму директиви #include, розглянемо програму, де оголошення функцій користувача max3( ) і fakt() – обчислення факторіалу, винесено в окремо створений заголовний файл з ім'ям u_head.h, а визначення функцій користувача max3( ) і fakt() – обчислення факторіалу, винесено в окремо створений файл з ім'ям u_head.cpp. Для цього додамо в проект новий файл з текстом програми та новий заголовний файл. Після цього вікно Solution Explorer проекту матиме такий вигляд (рис. 28).

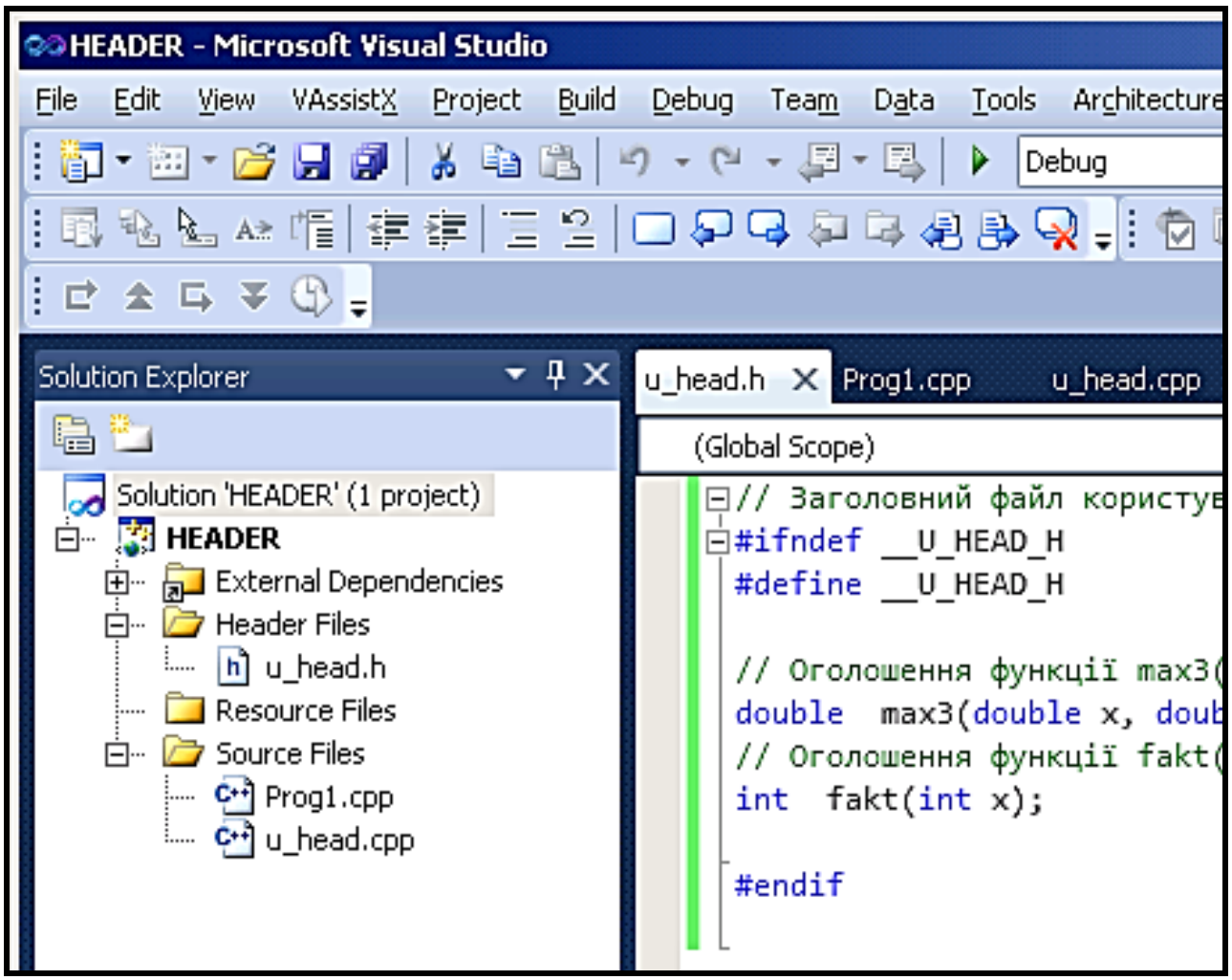


Рис. 28. **Проект з декількома файлами**

Заповнимо файли проекту необхідним змістом.

```
// Заголовний файл користувача  (u_head.h)
#ifndef __U_HEAD_H
#define __U_HEAD_H
    // Оголошення функції max3()
   double  max3(double x, double y, double z);
   // Оголошення функції fakt()
   int  fakt(int x);
#endif

// Файл користувача (u_head.cpp)
// Визначення функції max3()
double  max3(double x, double y, double z)
```

```
{
  double max=x;
  if (y > max)
    max = y;
  if (z > max)
   max = z;
  return max;
}

// Визначення функції fakt()
int  fakt(int x)
{
  int f = 1;
  for (int i=1; i<=x; i++)
    f *= i;                 // або  f = f * i;
  return f;
}

//  Програма 4.7
// Створення заголовного файлу користувача
// u_head.cpp для  визначення функцІй:
// max3() – пошуку максимального з трьох чисел І
// fakt() – розрахунок факторіалу аргументу
// МІсце оголошення функцій – заголовний файл користувача u_head.h
// МІсце визначення функцій – файл користувача u_head.cpp
// Виклик функції –  за іменем
// Повертане значення – використовується
// Метод передачі параметрів у функцію – за значенням

#include <iostream>
#include <locale.h>
#include <conio.h>
  // Підключення заголовного файлу користувача  u_head.h
  // Повний шлях вказувати не обов'язково
  // (відлік ведеться з поточної директорії bin)
#include "u_head.h"
using namespace std;
```

```
int main ()
{
  system("CLS");
  double  a, b, c;
  int d;
  setlocale(0,"RUS");
  cout << "\n ПОСЛІДОВНО ВВЕДІТЬ ТРИ ЧИСЛА: a,b,c:    " << endl;
  cin >> a;  cin >> b;  cin  >> c;
  cout << "\n ВИ ВВЕЛИ: a = " << a << "\t b = " << b << "\t з = " << c ;
  cout << "\n ТОМУ  max (a,b,c) = " << max3(a,b,c);
  cout << "\n ВВЕДІТЬ ОДНЕ ЦІЛЕ ЧИСЛО:  " << endl;
  cin >> d;
  cout << " \n ВИ ВВЕЛИ ЧИСЛО   " << d;
  cout  <<  "\n\n ОБЧИСЛЕННЯ ФАКТОРІАЛУ ЧИСЛА:  fakt(" << d << ") = "
<< fakt(d) <<endl;

  return 0;
}
```

Варіант результату роботи програми (рис. 29).

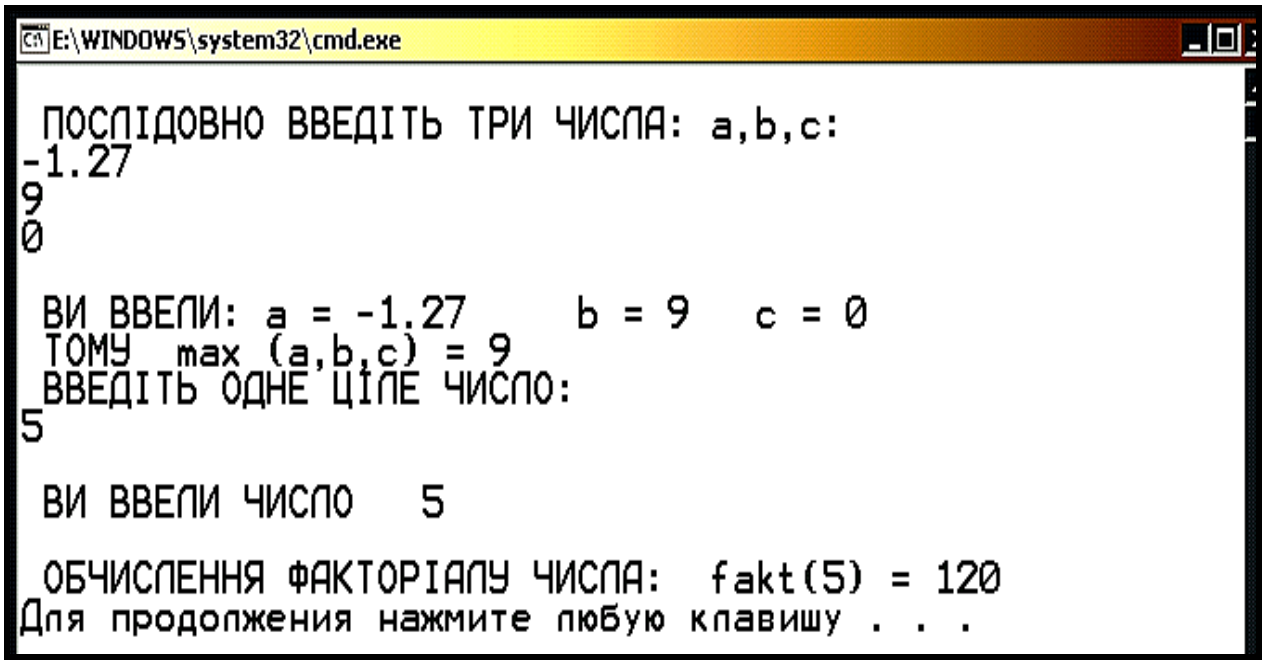


Рис. 29. **Приклад роботи багатофайлового проекту**

Завдання 1 (по варіантах).

Використовуючи механізм перевантаження функції, розробити і відладити програму обчислення значення = f(x, y, z) для різних типів параметрів. Конкретний варіант функції взяти із завдання 2 лабораторної роботи № 1. Передбачити завдання параметрів функції "за замовчуванням" і з клавіатури.

Як зразки використовувати фрагменти програм 4.1, 4.5, 4.6.

Завдання 2 (по варіантах).

Розробити і відладити програму обчислення виразу f(x), приведеного в додатку В.

Кожній з трьох гілок обчислення значення у повинна відповідати окрема функція (fun_1, fun_2 і fun_3). НА 12 БАЛІВ: Передача параметрів у функцію fun_1 повинна здійснюватися за значенням, у функцію fun_2 – використовуючи покажчики, а у функцію fun_3 – на базі посилань.

Як зразки використовувати фрагменти програм 4.3 і 4.4.

Завдання 3 (по варіантах).

Аналогічно завданню 2.2, проте всі оголошення функцій помістити в окремий заголовний файл з ім'ям ***.h, а визначення функцій помістити в окремий файл з ім'ям ***.cpp.

Як зразки використовувати фрагменти програм 4.3, 4.4 і 4.7.

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. Приведіть 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;
- узагальнені способи формування таблиць різної конфігурації за допомогою псевдосимволів.
- 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 в С++ – це набір однотипних даних (об'єктів), що мають загальне ім'я і що розрізняються місцеположенням в цьому наборі (або індексом, привласненому кожному елемента масиву).

У програмі можуть бути масиви цілих чисел або чисел з плаваючою точкою, символьні масиви і так далі.

Опис одновимірного масиву

Опис складається із специфікації типу, ідентифікатора і розмірності масиву (рис. 30).

Звернення до елементів масиву виконується за їх індексами, які завжди починаються від нуля. Якщо, наприклад, оголосити в програмі масив на ім'я imas з 100 цілих чисел

int imas[100];

то найперший елемент масиву отримує позначення imas[0], наступний, – imas[1] і так далі до останнього елемента imas [99].

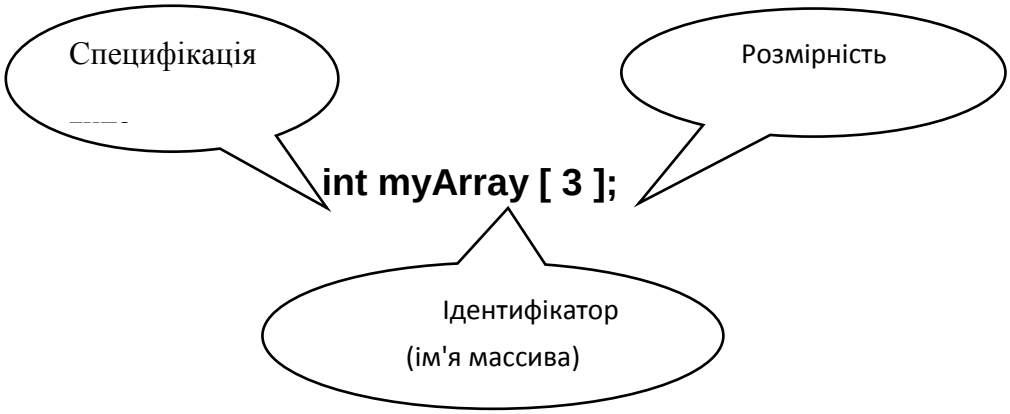


Рис. 30. **Ініціалізація масиву і доступ до елементів масиву**

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

```
char cSy[20];       //Масив з 20 символів
float fWe[16];      //Масив з 16 чисел з плаваючою точкою.
```

Після оголошення масиву його можна заповнити числами, використовуючи індексного оператора [ ], наприклад:

```
int    myArray [ 3 ];   // Оголошення масиву
myArray [ 0 ] = -12; // Ініціалізація першого елемента (його
                        // індекс дорівнює нулю) значенням –12
myArray [ 1 ] = 5;
myArray [ 2 ] = 37;
```

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

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

Масив можна оголосити і заповнити одночасно, наприклад:

```
int   myArray [ 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 myArray[ 5 ]= { 0, 1, 2 };   // myArray 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 і їх значення відомі заздалегідь, масив можна ініціалізувати разом з його оголошенням, уклавши перелік значень у фігурні дужки:

```
int  iNs[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 з 100 елементів, заповнених натуральним рядом чисел. Це робиться таким чином:

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

До елементів масиву можна звертатися і вибірково, наприклад

```
test[50]=0x7FFA;
```

Збереження одновимірних масивів

У C++ одновимірний масив логічно зберігається як послідовність впорядкованих елементів в пам'яті. Кожен елемент відноситься до одного і того ж типу даних. Наприклад, масив A розташовується в пам'яті таким чином (рис. 31):

Рис. 31. **Послідовне розташування елементів масиву в пам'яті**

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 місцеположенням в пам'яті першого елементу A[0]. Елементи A[1], A[2] і так далі слідують за ним послідовно.

Припустимо, що sizeof(type) = M, тоді весь масив A займає M * arraysize байтів (рис. 32).

Компілятор задає таблицю, названу дескриптором масиву (dope vector), для ведення запису характеристик масиву.

Таблиця включає інформацію про розмір кожного елемента масиву, початкову адресу масиву і кількість елементів в 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.

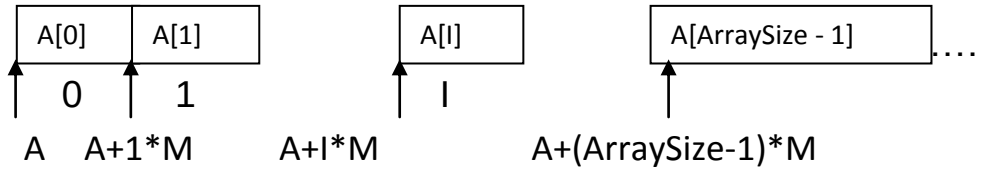


Рис. 32. **Адресація елементів масиву**

Приклад. Припустимо, що змінна типу float зберігається з використанням 4 байт (тобто sizeof(float) = 4) і масив Macizo[ ] починається в пам'яті за адресою 20000. Нехай

float Macizo[18];

Тоді елемент масиву Macizo[18] розміщується за адресою

20000 + 18*4 = 20072.

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 cSz[17]={'С','і','м','в','о', 'л','ь','н','ы', 'й'', ','м','а','с','с','и','в'};

Такий масив займає в пам'яті точно 17 байт.

У другому випадку масив ініціалізувався символьним рядком:

char cSz[20]="Символьный масив";

При такій ініціалізації компілятор записує в пам'ять в 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 доцільно лише в тих випадках, коли він заповнюватиметься програмно рядками різної довжини. Якщо ж даний текстовий рядок змінюватися не буде,

зручніше оголосити її без рекомендації довжини, залишивши в оголошенні пару квадратних дужок, які скажуть компілятору, що змінна cSz є не одиничним символом, а символьним масивом:

char cSz[]="Символьный масив";

Компілятор, виділяючи пам'ять під цей масив, сам визначить його довжину, яка в даному випадку дорівнюватиме 18 байтам.

Типові дії над масивами можна представити такою демонстраційною програмою:

```
#include <iostream>
#include <iomanip>
#include <locale.h>
#define N 10

using namespace std;

int main ()
{
  int i, j, m[N];
  system("CLS");
  setlocale(0,"RUS");

  // 1. Ініціалізація масиву нулями. ////////////////////////
  for (i = 0; i < N; i ++)   m [i] = 0;

  // 2. Ініціалізація масиву випадковими числами //
  for (i = 0; i < N; i ++)   m [i] = rand()%11;

  // 3. Друк масиву в рядок. ////////////////////////////////
  for (i = 0; i < N; i ++)  cout << setw(4) << m[i];
  cout << endl;

  // 4. Обчислення суми елементів масиву. ///////////////////
  int sum = 0;
  for (i = 0; i < N; i++)  sum += m[i];
  cout << "Сума =" << sum << endl;
```

```
 // 5. Виведення елементів масиву у вигляді гістограми.
  cout << "Елемент" << setw(13)
       << "Значення" << setw(13)
       << "Гістограма" << endl;
for (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 (i = 0; i < N; i ++)    cout << setw(4) << m[i];
   cout << endl;

 // сортування
  for ( i = 0; i<N-1; i++ )  // к-ть  проходів по масиву
     for ( 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 (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 (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 (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 порядку.
//перший спосіб перестановки
for ( i=0, j=N-1; i<j; ++i,--j )
  {
     temp = m[i];
     m[i]= m[j];
     m[j]= temp;
  }
//друк
  cout<< "Перестановка 1\n";
  for (i = 0; i < N; i ++) cout << setw(4) << m[i];  cout << endl;
//другий спосіб перестановки
for ( i = 0; i<(N/2); i++ )
  {
     temp    = m[i];
      m[i]=m[N-1-i];
      m[N-1-i]=temp;
  }
// друк
  cout<< "Перестановка 2\n";
  for (i = 0; i < N; i ++) cout << setw(4) << m[i];  cout << endl;

 // 10. Двійковий пошук у відсортованому масиві.///////////////
   int low, high, middle;
```

```
        flag = 0;
        low = 0;
        high = N-1;
        cout << "Введіть ключ"  << endl;
        cin >> key;
// пошук
  while ( (low <= high) && !flag)
    {
      middle = ( low + high ) / 2;
      if ( key == m[middle])  flag = 1;
         else if ( key < m[middle])
                    high = middle-1;
              else low = middle+1;
    } // while
    if ( flag )  cout << "Шуканий елемент має індекс=" << middle << endl;
       else  cout << "Элемент не знайдений" << endl;
  return 0;
}
```

Результат роботи програми наведений на рис. 33.

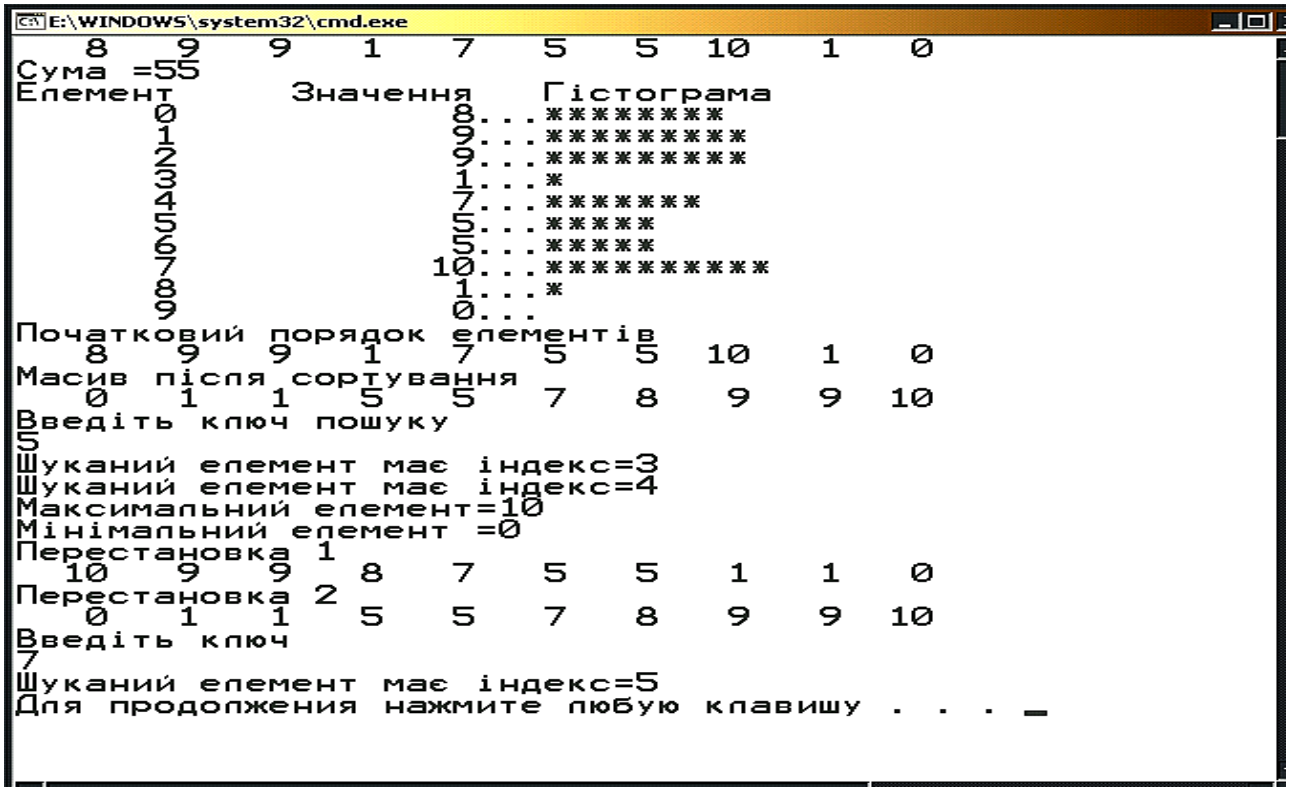


Рис. 33. **Результат роботи демонстраційної програми з масивом**

При відладці програм часто буває зручним ініціювати масив даними (числами або символами), що вибираються випадковим чином, наприклад як у кроці два наведеного прикладу. Для цього можна використовувати спеціальні функції з бібліопапки stdlib.h. Розглянемо їх докладніше.

Функція rand() генерує псевдовипадкові числа в діапазоні 0 .. RAND_MAX, де RAND_MAX – константа, зазвичай співпадаюча з максимальним цілим значенням для даної реалізації.

Функція random (int num) генерує випадкові числа в діапазоні 0 .. num – 1.

Функція randomize() ініціює генератор випадкових чисел від таймера.

Функція srand (unsigned seed) ініціює генератор випадкових чисел так, що послідовність, що генерується, залежить від аргументу; якщо аргумент цієї функції не змінювати, то завжди генеруватиметься одна і та ж послідовність.

Приклади:

1. int i = 1 + rand() % 6; // генеруються вип. числа в діапазоні 1.. 6
2. i = rand() & 2; // генеруються вип. числа в діапазоні 0..1

Завдання 1. Скласти програму яка виконує обробку масива у функції та реалізує такі дії:

Варіант 1

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) суму негативних елементів масиву;

2) добуток елементів масиву, розташованих між максимальним і мінімальним елементами.

Упорядкувати елементи масиву за збільшенням.

Варіант 2

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) суму позитивних елементів масиву;

2) добуток елементів масиву, розташованих між максимальним за модулем і мінімальним за модулем елементами.

Упорядкувати елементи масиву за спаданням.

Варіант 3

У одновимірному масиві, що складається з n цілих елементів, обчислити:

1) добуток елементів масиву з парними номерами;

2) суму елементів масиву, розташованих між першим і останнім нульовими елементами.

Перетворити масив так, щоб спочатку розташовувалися всі позитивні елементи, а потім – всі негативні (елементи, рівні 0, вважати за позитивні).

Варіант 4

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) суму елементів масиву з непарними номерами;

2) суму елементів масиву, розташованих між першим і останнім негативними елементами.

Стискувати масив, видаливши з нього всі елементи, модуль яких не перевищує 1. Елементи, що звільнилися в кінці масиву, заповнити нулями.

Варіант 5

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) максимальний елемент масиву;

2) суму елементів масиву, розташованих до останнього позитивного елемента.

Стискувати масив, видаливши з нього всі елементи, модуль яких знаходиться в інтервалі [a, b]. Елементи, що звільнилися в кінці масиву, заповнити нулями.

Варіант 6

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) мінімальний елемент масиву;

2) суму елементів масиву, розташованих між першим і останнім позитивними елементами.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, рівні нулю, а потім – всі інші.

Варіант 7

У одновимірному масиві, що складається з n цілих елементів, обчислити:

1) номер максимального елемента масиву;

2) добуток елементів масиву, розташованих між першим і другим нульовими елементами.

Перетворити масив так, щоб в першій його половині розташовувалися елементи, що стояли в непарних позиціях, а в другій половині – елементи, що стояли в парних позиціях.

Варіант 8

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) номер мінімального елемента масиву;

2) суму елементів масиву, розташованих між першим і другим негативними елементами.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, модуль яких не перевищує 1, а потім – всі інші.

Варіант 9

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) максимальний по модулю елемент масиву;

2) суму елементів масиву, розташованих між першим і другим позитивними елементами.

Перетворити масив так, щоб елементи, рівні нулю, розташовувалися після всіх інших.

Варіант 10

У одновимірному масиві, що складається з n цілих елементів, обчислити:

1) мінімальний по модулю елемент масиву;

2) суму модулів елементів масиву, розташованих після першого елементу, рівного нулю.

Перетворити масив так, щоб в першій його половині розташовувалися елементи, що стояли в парних позиціях, а в другій половині – елементи, що стояли в непарних позиціях.

Варіант 11

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) номер мінімального по модулю елемента масиву;

2) суму модулів елементів масиву, розташованих після першого негативного елемента.

Стискувати масив, видаливши з нього всі елементи, величина яких знаходиться в інтервалі [a, b]. Елементи, що звільнилися в кінці масиву, заповнити нулями.

Варіант 12

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) номер максимального по модулю елемента масиву;

2) суму елементів масиву, розташованих після першого позитивного елемента.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, ціла частина яких лежить в інтервалі [a, b], а потім – всі інші.

Варіант 13

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість елементів масиву, лежачих в діапазоні від A до B;

2) суму елементів масиву, розташованих після максимального елемента.

Упорядкувати елементи масиву за спаданням модулів елементів.

Варіант 14

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість елементів масиву, рівних 0;

2) суму елементів масиву, розташованих після мінімального елемента.

Упорядкувати елементи масиву за збільшенням модулів елементів.

Варіант 15

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість елементів масиву, великих 3;

2) добуток елементів масиву, розташованих після максимального по модулю елемента.

Перетворити масив так, щоб спочатку розташовувалися всі негативні елементи, а потім – всі позитивні (елементи, рівні 0, вважати за позитивні).

Варіант 16

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість негативних елементів масиву;

2) суму модулів елементів масиву, розташованих після мінімального по модулю елемента.

Замінити всі негативні елементи масиву їх квадратами і впорядкувати елементи масиву за збільшенням.

Варіант 17

У одновимірному масиві, що складається з n цілих елементів, обчислити:

1) кількість позитивних елементів масиву;

2) суму елементів масиву, розташованих після останнього елемента, рівного нулю.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, ціла частина яких не перевищує 1, а потім – всі інші.

Варіант 18

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) кількість елементів масиву, менших 3;

2) суму цілих частин елементів масиву, розташованих після останнього негативного елемента.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, що відрізняються від максимального не більше ніж на 20 %, а потім – всі інші.

Варіант 19

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) добуток негативних елементів масиву;

2) суму позитивних елементів масиву, розташованих до максимального елемента.

Змінити порядок проходження елементів в масиві на зворотний.

Варіант 20

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

1) добуток позитивних елементів масиву;

2) суму елементів масиву, розташованих до мінімального елемента.

Упорядкувати за збільшенням окремо елементи, що стоять на парних місцях, і елементи, що стоять нанепарних місцях.

Завдання 2. Підвищеної складності (на оцінку 11 – 12 балів)

Варіант 1. Послідовність з десяти цілих чисел (як позитивних, так і негативних) представлена у вигляді одновимірного масиву. Знайти і вивести на екран підпослідовність підряд чисел, що йдуть, сума яких максимальна. Наприклад, для послідовності : 1 -8 3 2 -1 4 -6 2 1 -5 відповіддю буде така підпослідовність: 3 2 -1 4

Варіант 2. Дана цілочисельна послідовність (одновимірний масив цілих чисел). Написати програму знаходження кількості різних елементів цієї послідовності. Вивести всі елементи, що зустрічаються більше одного разу.

Варіант 3. У заданій послідовності чисел довжиною n (n<100) визначити довжину найбільшої впорядкованої за збільшенням підпослідовності сусідніх елементів.

Контрольні питання

1. Дайте визначення масиву.
2. Як розташовується масив в 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 елементу масиву?

Лабораторна робота № 6

Підготовка і розв’язання на ПЕОМ задач обробки двовимірних масивів

Мета лабораторної роботи – освоєння методики обробки інформації в двовимірних масивах.

Перед виконанням лабораторної роботи студент повинен знати:

- визначення і правила опису двовимірних масивів різних типів;
- способи доступу до елементів масиву даних;
- способи ініціалізації елементів масиву;
- способи алгоритмізації пошуку мінімальних і максимальних значень в двовимірних масивах;

Після виконання лабораторної роботи студент повинен вміти:

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

Короткі теоретичні відомості

Двовимірні масиви

Двовимірний масив є структурованим типом даних, який створюється шляхом вкладення одновимірних масивів.

Доступ до елементів виконується за індексами рядків і стовпців.

У С++ оголошення двовимірного масиву визначає кількість рядків, кількість стовпців і тип даних елементів масиву. Наприклад, масив T

type T [RowCount] [ColCount] ;

може бути представлений таким чином:

		Стовпці										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Рядки	**0**											
	1			10								
	2							-3				
	3											

Посилання на елементи масиву T проводиться за допомогою індексів рядка і стовпця:

T[i] [j] 0 <= i <= RowCount – 1 0 <= j <= ColCount – 1

Наприклад, матриця T – це масив цілих розміром 4 x 11:

int T [4] [11];

При чому значення T[1] [2]= 10 і T[2] [6]= -3.

Концепція двовимірного масиву може бути розширена для обхвату основних багатовимірних масивів, елементи в яких доступні за допомогою трьох або більші за індекси.

Двовимірний масив можна представити як список одновимірних масивів.

Наприклад, T[0] – це рядок 0, яка складається з ColumnCount окремих елементів.

Запис int T[ ] [11];

вказує, що T є списком з 11-ти елементних масивів.

Двовимірний масив може ініціалізуватися привласненням елементам по одному рядку кожного разу. Наприклад, масив T задає таблицю розміром 3 x 4:

int T [3] [4] = { {20, 5 -30, 0}, {-40, 15, 100, 80}, {3, 0, 0 -1} };

Двовимірні таблиці є найбільш поширеною формою представлення даних економічного характеру. Будь-яка таблиця – документ, як правило, містить:

- найменування, так звану "шапку" з іменами колонок (рядків);
- рядки – записи таблиці, що мають однакову довжину, і що містять декілька різнотипних за характером, але зв'язаних між собою полів – реквізитів;
- підсумкову частину.

Багатовимірний масив відповідно до синтаксису мови є масив масивів, тобто масив, елементами якого служать масиви. Визначення багатовимірного масиву в загальному випадку повинне містити відомості про тип, розмірність і кількість елементів кожної розмірності:

type ім'я_масиву [К1][К2]...[KN];

Тут type – допустимий тип (основний або похідний), ім'я масиву – ідентифікатор, N – розмірність масиву, К1 – кількість в масиві елементів розмірності N -1 кожен і так далі. Наприклад:

int ARRAY[4][3][б];

Тривимірний масив array складається з чотирьох елементів, кожен з яких – двомірний масив з розмірами 3 на 6. У пам'яті масив array розміщується в порядку зростання найправішого індексу, тобто наймолодша адреса має елемент array [0] [0] [0], потім йде array [0] [0] [1] і так далі.

З урахуванням порядку розташування в пам'яті елементів багатовимірного масиву потрібно розміщувати початкові значення його елементів і в списку ініціалізації (поправка на правостороннє написання фраз і слів в європейських мовах на відміну від напряму зростання адрес абсолютно природна).

int ARRAY[4][3][6]= { 0, 1, 2, 3, 4, 5, 6, 7 };

У даному визначенні початкові значення отримали тільки "перші" 8 елементів тривимірного масиву, а саме:

ARRAY[0][0][0] = 0
ARRAY[0][0][1] = 1
ARRAY[0][0][2] = 2
ARRAY[0][0][3] = 3
ARRAY[0][0][4] = 4
ARRAY[0][0][5] = 5
ARRAY[0][1][0] = 6
ARRAY[0][1][1] = 7

Решта елементів масиву array залишилася не ініціалізованими і вони набудуть початкових значень відповідно до статусу масиву.

Якщо необхідно ініціалізувати тільки частину елементів багатовимірного масиву, але вони розміщені не на його початку або не підряд, то можна вводити додаткові фігурні дужки, кожна пара яких виділяє послідовність значень, що відносяться до однієї розмірності (не можна використовувати дужки без інформації всередені них).

Наступне визначення з ініціалізацією тривимірного масиву

int A[4][5][6]= { { {0} },
{ {100}, {110, 111} },
{ {200}, {210}, {220, 221, 222} };

так задає тільки деякі значення його елементів:

A[0][0][0] =0
A[1][0][0]=100 A[1][1][0] =110 A[1][1][1]=111
A[2][0][0]=200 A[2][1][0] = 210
A[2][2][0]=220 A[2][2] [1] = 221 A[2] [2] [2] = 222

Решта елементів масиву явно не ініціалізувалася. Якщо багато вимірний масив при визначенні ініціалізувався, то його найлівіша розмірність може в дужках не вказуватися. Кількість елементів компілятор визначає за числом членів в ініціалізуючому списку. Наприклад, визначення

float matr [ ] [5] ={ {1}, {2}, {3} };

формує масив matr з розмірами 3 на 5, але не визначає явно початкових значень всіх його елементів. Оператор

cout << "\nsizeof(matr)= " << sizeof (matr);

виведе на екран: sizeof (matrix) = 60

Початкові значення отримують тільки

matr[0][0]= 1

matr[1][0]= 2

matr[2][0]= 3

Методи обробки даних, розміщених в двовимірних і багатовимірних масивах аналогічні методам роботи з одновимірними масивами.

Розглянемо приклад рішення задачі обробки даних, розміщених в двовимірному масиві.

Приклад 1

Написати програму, яка визначає номер рядка квадратної матриці, сума елементів якої максимальна.

```
// Рядок з максимальною сумою елементів
#include<stdio.h>
#include <conio.h>
#include <locale.h>

#define N   3     // розмір квадратної матриці

void main ()
{
    int m[N][N+1]; // останній стовпець використовуваний
                   // для зберігання суми елементів рядка
    int max;        // рядок з максимальною сумою елементів
    int i,j;        // індекси матриці

    setlocale(0,"RUS");
    puts   ( "\nВизначення рядка з максимальною сумою елементів \n" ) ;
    printf  ("Введіть матрицю  %i x %i \n",  N,  N) ;
    for  (i = 0;   i < N;   i++)
    {
        printf  ( " Елементи % i-го рядка -> ", i+1);
        for   ( j = 0;   j < N;   j++)
            scanf (  " %i",   &m[i][j]); }
```

```
    // для кожного рядка обчислимо суму елементів
    for (i=0;i<N;i++){
        m [i][N] = 0;
        for(j=0;  j<N;  j++)
            m[i][N] += m[i][j];
    }
    // Виведемо матрицю та суму на екран
    for(i=0; i<N; i++){
        for(j=0; j<=N; j++){
            printf("%s%3d",(j==N)?" Сума= ":" ",m[i][j]);
        }
        printf("\n");
    }
    // знайдемо рядок з максимальною сумою
    max = 0;
    for  ( i=1; i<N; i++)
        if ( m[i][N]>m[max][N])max = i;
    printf   ( " \n В % i-й рядку сума елементів", max+1);
    printf   ( " максимальна і дорівнює  %d \n ", m[max][N]);
    printf   ( " \n Для завершення натисніть <Enter> \n " );
    getch();
}
```

Результат роботи програми наведено на рис. 34.

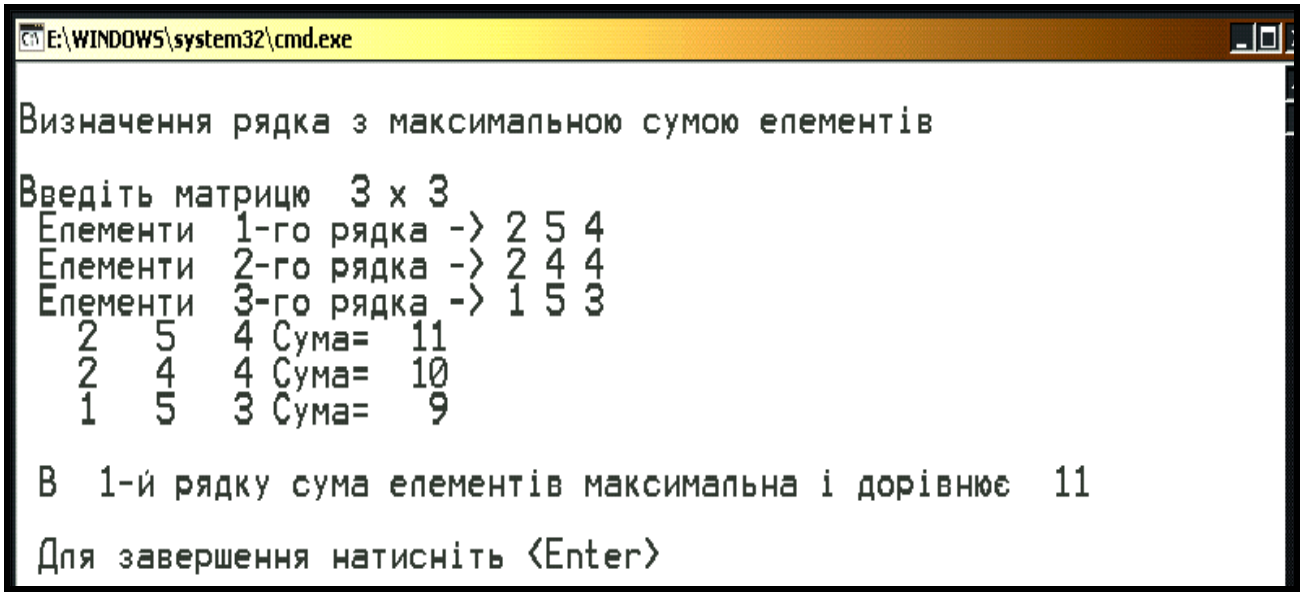


Рис. 34. **Робота програми з пошуку рядка з максимальною сумою**

Завдання 1. Скласти програму обробки двовимірних масивів згідно з таких варіантів. Обробку масиву виконати у функції.

Варіант 1

Дано цілочисельну прямокутну матрицю. Визначити:

кількість рядків, що не містять жодного нульового елементу;

максимальне з чисел, що зустрічаються в заданій матриці більше одного разу.

Варіант 2

Дано цілочисельну прямокутну матрицю. Визначити кількість стовпців, що не містять жодного нульового елементу.

Характеристикою рядка цілочисельної матриці назвемо суму її позитивних парних елементів. Переставляючи рядки заданої матриці, розташувати їх відповідно до зростання характеристик.

Варіант 3

Дано цілочисельну прямокутну матрицю. Визначити:

кількість стовпців, що містять хоч би один нульовий елемент;

номер рядка, в якому знаходиться щонайдовша серія однакових елементів.

Варіант 4

Дано цілочисельну квадратну матрицю. Визначити:

добуток елементів в тих рядках, які не містять негативних елементів;

максимум серед сум елементів діагоналей, паралельних головній діагоналі матриці.

Варіант 5

Дано цілочисельну квадратну матрицю. Визначити:

суму елементів в тих стовпцях, які не містять негативних елементів;

мінімум серед сум модулів елементів діагоналей, паралельних побічній діагоналі матриці.

Варіант 6

Дано цілочисельну прямокутну матрицю. Визначити:

суму елементів в тих рядках, які містять хоч би один негативний елемент;

номери рядків і стовпців всіх сідлових точок матриці.

Примітка. Матриця A має сідлову точку Aij, якщо Aij є мінімальним елементом в i-й рядку і максимальним в j-м стовпці.

Варіант 7

Для заданої матриці розміром 8 на 8 знайти такі k, що k-й рядок матриці збігається з k-м стовпцем.

Знайти суму елементів в тих рядках, які містять хоч би один негативний елемент.

Варіант 8

Характеристикою стовпця цілочисельної матриці назвемо суму модулів його негативних непарних елементів. Переставляючи стовпці заданої матриці, розташувати їх відповідно до зростання характеристик.

Знайти суму елементів в тих стовпцях, які містять хоч би один негативний елемент.

Варіант 9

Сусідами елементу Aij в матриці назвемо елементи Arc, де i - 1 < r < i + 1, j - 1 < c < j+1. (r, c) != (i, j). Операція згладжування матриці дає нову матрицю того ж розміру, кожен елемент якої виходить як середнє арифметичне наявних сусідів відповідного елементу початкової матриці. Побудувати результат згладжування заданої дійсної матриці розміром 10 на 10. У згладженій матриці знайти суму модулів елементів, розташованих нижче за головну діагональ.

Варіант 10

Елемент матриці називається локальним мінімумом, якщо він строго менше всіх сусідів, які є у нього. Підрахувати кількість локальних мінімумів заданої матриці розміром 10 на 10. Знайти суму модулів елементів, розташованих вище за головну діагональ.

Варіант 11

Коефіцієнти системи лінійних рівнянь задані у вигляді прямокутної матриці. За допомогою допустимих перетворень привести систему до трикутного вигляду. Знайти кількість рядків, середнє арифметичне елементів яких менше заданої величини.

Варіант 12

Ущільнити задану матрицю, видаляючи з неї рядки і стовпці, заповнені нулями. Знайти номер першою з рядків, що містять хоч би один позитивний елемент.

Варіант 13

Здійснити циклічний зсув елементів прямокутної матриці на n елементів управо або вниз (залежно від введеного режиму), n може бути більше кількості елементів в рядку або стовпці.

Варіант 14

Здійснити циклічний зсув елементів квадратної матриці розмірності M x N управо на k елементів таким чином: елементи 1-го рядка зрушуються в останній стовпець зверху вниз, з нього – в останній рядок справа наліво, з неї – в перший стовпець від низу до верху, з нього – в перший рядок; для решти елементів – аналогічно.

Варіант 15

Дано цілочисельну прямокутну матрицю. Визначити номер першого із стовпців, що містять хоч би один нульовий елемент.

Характеристикою рядка цілочисельної матриці назвемо суму її негативних парних елементів. Переставляючи рядки заданої матриці, розташувати їх відповідно до спадання характеристик.

Варіант 16

Упорядкувати рядки цілочисельної прямокутної матриці за збільшенням кількості однакових елементів в кожному рядку.

Знайти номер першого із стовпців, що не містять жодного негативного елемента.

Варіант 17

Шляхом перестановки елементів квадратної дійсної матриці добитися того, щоб її максимальний елемент знаходився в лівому верхньому кутку, наступний по величині – в позиції (2,2), наступний по величині – в позиції (3,3) і т. д., заповнивши таким чином всю головну діагональ.

Знайти номер першою з рядків, що не містять жодного позитивного елемента.

Варіант 18

Дано цілочисельну прямокутну матрицю. Визначити:

кількість рядків, що містять хоч би один нульовий елемент;

номер стовпця, в якому знаходиться щонайдовша серія однакових елементів.

Варіант 19

Дана цілочисельна квадратна матриця. Визначити:

суму елементів в тих рядках, які не містять негативних елементів;

мінімум серед сум елементів діагоналей, паралельних головній діагоналі матриці.

Варіант 20

Дано цілочисельну прямокутну матрицю. Визначити:

кількість негативних елементів в тих рядках, які містять хоч би один нульовий елемент;

номери рядків і стовпців всіх сідлових точок матриці.

Примітка. Матриця А має сідлову точку Аij, якщо Аij є мінімальним елементом в і-й рядку і максимальним в j-м стовпці.

Завдання 2: Підвищеній складності (на оцінку 11 – 12 балів).

Варіант 1. Перестановкою рядків і стовпців упорядкувати за збільшенням елементи головної діагоналі квадратної матриці.

Варіант 2. Написати програму циклічного зсуву елементів квадратної матриці на задану кількість позицій, вважаючи, що матриця зв'язана за рядками.

Варіант 3. Написати програму, яка для квадратної матриці визначає максимум серед сум елементів, розташованих на лініях, паралельних головній діагоналі, і вище.

Варіант 4. Дано цілочисельну прямокутну матрицю. Переходом назвемо переміщення до сусіднього елементу по горизонталі або по вертикалі. Знайти кількість різних шляхів проходу матриці з лівого верхнього кута в правий нижній, якщо допустимими є тільки переходи "праворуч на один елемент" і "вниз на один елемент". Визначити шлях, сума елементів якого максимальна.

Контрольні питання

1. Дайте визначення масиву.
2. Що розуміється під двовимірним масивом?
3. Як розташовується двомірний масив в пам'яті?
4. Вкажіть приклади завдання розмірності масиву.
5. Як можна ініціалізувати елементи масиву?
6. Приведіть приклад визначення двовимірного масиву.
7. Яке інформаційне навантаження несе ім'я одновимірного масиву?
8. Яке інформаційне навантаження несе ім'я двомірного масиву?
9. Яка операція використовується для визначення адреси довільного елемента масиву?

Лабораторна робота № 7

Підготовка і розв’язання на ПЕОМ задач обробки масивів з використанням покажчиків

Мета лабораторної роботи – освоїти основні прийоми використання покажчиків в програмах мовою С++.

Перед виконанням лабораторної роботи студент повинен знати:

- поняття покажчика і основні операції з покажчиками;
- взаємозв'язок між покажчиками і масивами;
- основи динамічного розподілу пам'яті.

Після виконання лабораторної роботи студент повинен вміти: використовувати покажчики при розробці програм на мові С++.

Теоретичний матеріал

Покажчики

Коли компілятор обробляє оператора визначення змінної, наприклад, int і =10, він виділяє пам'ять відповідно до типу (int) і ініціалізує її вказаним значенням (10). Усі звернення в програмі до змінної за її іменем (і) замінюються компілятором на адресу області пам'яті, в якій зберігається значення змінної. Програміст може визначити власні змінні для зберігання адрес областей пам'яті. Такі змінні нази-ваються покажчиками. Отже, *покажчики призначені для зберігання адрес областей пам'яті.* У С++ розрізняють три види покажчиків – покажчики на об'єкт, на функцію і на void, що відрізняються властивостями і набором допустимих операцій. Покажчик не є самостійним типом, він завжди пов'язаний з яким-небудь іншим конкретним типом.

Покажчик на функцію містить адресу в сегменті коду, за яким розташовується виконуваний код функції, тобто адреса, за якою передається управління при виклику функції. Покажчики на функції використовуються для непрямого виклику функції (не через її ім'я, а через звернення до змінної, що зберігає її адресу), а також для передачі імені функції в іншу функцію як параметр. Покажчик функції має тип

покажчик функції, що повертає значення заданого типа і що має аргументи заданого типа:

тип (*ім'я) (список_типів_аргументів);

Наприклад, оголошення:

int (*fun) (double, double);

задає покажчик з ім'ям fun на функцію, що повертає значення типу int і що має два аргументи типу double.

Покажчик на об'єкт містить адресу області пам'яті, в якій зберігаються дані певного типу (основного або складеного). Просте оголошення покажчика на об'єкт (надалі званого просто покажчиком) має вигляд:

тип *ім'я;

де тип може бути будь-яким, окрім посилання і бітового поля, при чому тип може бути до цього моменту тільки оголошений, але ще не визначений (отже, в структурі, наприклад, може бути присутнім покажчик на структуру того ж типу).

Зірочка відноситься безпосередньо до імені, тому для того, щоб оголосити декілька покажчиків, потрібно ставити її перед ім'ям кожного з них. Наприклад, в операторі

int *a, b *c;

описуються два покажчики на ціле з іменами a, c, а також ціла змінна b.

Розмір покажчика залежить від моделі пам'яті. Можна визначити покажчик на покажчик і т. д.

Покажчик на void застосовується в тих випадках, коли конкретний тип об'єкту, адресу якого потрібно зберігати, не визначений (наприклад, якщо в одній і тій же змінній в різні моменти часу потрібно зберігати адреси об'єктів різних типів).

Покажчику на void можна привласнити значення покажчика будь-якого типу, а також порівнювати його з будь-якими покажчиками, але перед виконанням яких-небудь дій з областю пам'яті, на яку він посилається, потрібно перетворити його до конкретного типу явним чином.

Покажчик може бути константою або змінною, а також вказувати на константу або змінну. Розглянемо приклади:

```
int i:                          // ціла змінна
const int ci = 1;               // ціла константа
int * pi:                        // покажчик на цілу змінну
const int * pci;                 // покажчик на цілу константу
int * const cp = &i;             // покажчик-константа на цілу змінну
const int * const cpc =   &ci;   // покажчик-константа на цілу константу
```

Як видно з прикладів, модифікатор const, що знаходиться між ім'ям покажчика і зірочкою, відноситься до самого покажчика і забороняє його зміну а const зліва від зірочки задає постійність значення, на яке він вказує. Для ініціалізації покажчиків використана операція отримання адреси &. Величини типу покажчик підкоряються загальним правилам визначення області дії, видимості і часу життя.

Ініціалізація покажчиків

Покажчики найчастіше використовують при роботі з динамічною пам'яттю, званою деякими естетами купою (переклад з англійської мови слова heap). Це вільна пам'ять, в якій можна під час виконання програми виділяти місце відповідно до потреб. Доступ до виділених ділянок динамічної пам'яті, званих динамічними *змінними, проводиться* тільки через покажчики. Час життя динамічних змінних – від точки створення до кінця програми або до явного звільнення пам'яті. У C++ використовується два способи роботи з динамічною пам'яттю. Перший використовує сімейство функцій malloc і дістався у спадок від C, другий використовує операції new і delete.

При визначенні покажчика треба прагнути виконати його ініціалізацію, тобто привласнення початкового значення. Ненавмисне використання неініціалізованих покажчиків – поширене джерело помилок в програмах. Ініціалізація записується після імені покажчика або в круглих дужках, або після знаку рівності.

Існують такі способи ініціалізації покажчика:

1. Привласнення покажчику адреси існуючого об'єкту:

за допомогою операції отримання адреси:

```
int a = 5;       // ціла змінна
int *p = &a;     // в покажчик записується адреса змінної n
int *p(&a);      // те ж саме іншим способом
```

за допомогою значення іншого покажчика, що ініціалізований:

```
int *r = p;
```

за допомогою імені масиву або функції, які трактуються як адреса:

```
int b[10];                    // масив
int *t = b;                   // привласнення адреси початку масиву
void f(int a ) { /* ... */ }  // визначення функції
void (*pf)(int);              // покажчик на функцію
pf = f;                       // привласнення адреси функції
```

2. Привласнення покажчику адреси області пам'яті в явному вигляді:

```
char* vp = (char *)0xB8000000;
```

Тут 0xB8000000 – шістнадцятерічна константа, (char *) – операція приведення типу: константа перетвориться до типу покажчик на char.

3. Привласнення порожнього значення:

```
int* suxx = NULL;
int* rulez = 0;
```

У першому рядку використовується константа NULL, визначена в деяких заголовних файлах C як покажчик, рівний нулю. Рекомендується використовувати просто 0, оскільки це значення типу int буде правильно перетворено стандартними способами відповідно до контексту. Оскільки гарантується, що об'єктів з нульовою адресою немає, порожній покажчик можна використовувати для перевірки того, чи посилається покажчик на конкретний об'єкт, чи ні.

4. Виділення ділянки динамічної пам'яті і привласнення її адреси покажчику:

за допомогою операції new:

```
int* n = new int;          // 1
int* m = new int (10);  // 2
```

```
int* q = new int [10];                    // 3
   за допомогою функції malloc:
int* u = (int *)malloc(sizeof(int));      // 4
```

У операторі 1 операцію new виконує виділення достатнього для розміщення величини типу int ділянки динамічної пам'яті і записує адресу початку цієї ділянки в змінну n. Пам'ять під саму змінну n (розміру, достатнього для розміщення покажчика) виділяється на етапі компіляції.

У операторі 2, окрім описаних вище дій, проводиться ініціалізація виділеної динамічної пам'яті значенням 10.

У операторі 3 операція new виконує виділення пам'яті під 10 величин типу int (масиву з 10 елементів) і записує адресу початку цієї ділянки в змінну q, яка може трактуватися як ім'я масиву. Через ім'я можна звертатися до будь-якого елементу масиву. Якщо пам'ять виділити не вдалося, за стандартом повинне породжуватися виключення bad_alloc. Старі версії компіляторів можуть повертати 0.

У операторі 4 робиться те ж саме, що і в операторі 1, але за допомогою функції виділення пам'яті malloc, успадкованої з бібліотеки C. У функцію передається один параметр – кількість пам'яті, що виділяється, в байтах. Конструкція (int*) використовується для приведення типу покажчика, що повертається функцією, до необхідного типу. Якщо пам'ять виділити не вдалося, функція повертає 0.

Рекомендується переважно використовувати операцію new, чим функцію malloc, особливо при роботі з об'єктами.

Звільнення пам'яті, виділеної за допомогою операції new, повинно виконуватися за допомогою delete, а пам'яті, виділеною функцією malloc – за допомогою функції free. При цьому змінна-покажчик зберігається і може ініціалізуватися повторно. Наведені динамічні змінні знищуються таким чином:

```
delete n; delete m; delete [] q; free (u);
```

Якщо пам'ять виділялася з допомогою new[], для звільнення пам'яті необхідно застосовувати delete[]. Розмірність масиву при цьому не

вказується. Якщо квадратних дужок немає, то ніякого повідомлення про помилку не видається, але помічений як вільний буде тільки перший елемент масиву, а інші опиняться недоступні для подальших операцій. Такі елементи пам'яті називаються сміттям.

За допомогою комбінацій зірочок, круглих і квадратних дужок можна описувати складені типи і покажчики на складені типи, наприклад, в операторі

```
int *(*p[10])();
```

оголошується масив з 10 покажчиків на функції без параметрів, що повертають покажчики на int.

За замовчуванням квадратні і круглі дужки мають однаковий пріоритет, більший, ніж зірочка, і розглядаються зліва направо. Для зміни порядку розгляду використовуються круглі дужки.

При інтерпретації складних описів необхідно дотримуватися правила з середини назовні:

1) якщо праворуч від імені є квадратні дужки, це масив, якщо дужки круглі – це функція;

2) якщо зліва є зірочка – це покажчик на проінтерпретовану раніше конструкцію;

3) якщо праворуч зустрічається закриваюча кругла дужка, необхідно використати наведені вище правила усередині дужок, а потім переходити назовні;

4) в останню чергу інтерпретується специфікатор типу.

Операції з покажчиками

З покажчиками можна виконувати такі операції: розадресація, або непряме звернення до об'єкту (*), привласнення, складання з константою, віднімання, інкремент (++), декремент (--), порівняння, приведення типів. При роботі з покажчиками часто використовується операція отримання адреси (&).

Операція розадресації**,** або розіменування, призначена для доступу до величини, адреса якої зберігається в покажчику. Цю операцію можна

використовувати як для отримання, так і для зміни значення величини (якщо вона не оголошена як константа):

```
char a;              // змінна типу char
char * p = new char;      /* виділення пам'яті під покажчик і під динамічну змінну типу char */
*p = 'A'; a = *p;              // привласнення значення обом змінним
```

Як видно з прикладу, конструкцію *ім'я_покажчика можна використовувати в лівій частині оператора привласнення, оскільки вона є L-значенням, тобто визначає адресу області пам'яті. Для простоти цю конструкцію можна вважати ім'ям змінної, на яку посилається покажчик. З нею допустимі всі дії, визначені для величин відповідного типу (якщо покажчик ініціалізував). На одну і ту ж область пам'яті може посилатися декілька покажчиків різного типа. Застосована ним операція розадресації дасть різні результати. Наприклад, програма

```
#include <stdio.h>
int main()
{
   unsigned long int A = 0Xcc77ffaa;
   unsigned short int* pint = (unsigned short int*) &A;
   unsigned char*    pchar = (unsigned char *) &A;
   printf(" | %x | %X | %X |\n", A, *pint, *pchar);
   return 0;
}
```

виведе на екран рядок (рис. 35):

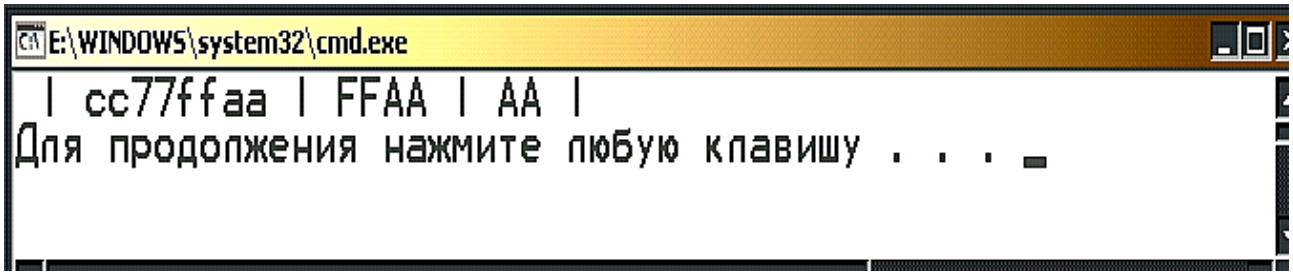


Рис. 35. **Ілюстрація програми роботи з покажчиками**

Значення покажчиків pint і pchar однакові, але розадресація **pchar** дає в результаті один молодший байт за цією адресою, а pint – два молодші байти. У наведеному прикладі при ініціалізації покажчиків були використані операції приведення типів. Синтаксис операції явного приведення типу простий: перед ім'ям змінної в дужках вказується тип, до якого її потрібно перетворити. При цьому не гарантується збереження інформації, тому в загальному випадку явних перетворень типу слід уникати.

При змішуванні у виразі покажчиків різних типів явне перетворення типів потрібне для всіх покажчиків, окрім void*. Покажчик може неявно перетворюватися в значення типу bool (наприклад, у виразі умовного оператора), при цьому ненульовий покажчик перетвориться в true, а нульовий в false. Привласнення без явного приведення типів допускається в двох випадках:

покажчикам типу void*;

якщо тип покажчиків справа і зліва від операції привласнення один і той же.

Таким чином, неявне перетворення виконується тільки до типу void*. Значення 0 неявно перетвориться до покажчика на будь-який тип. Привласнення покажчиків на об'єкти покажчикам на функції (і навпаки) неприпустимо. Заборонено також привласнювати значення покажчикам-константам, втім, як і константам будь-якого типа (привласнювати значення покажчикам на константу і змінним, на які посилається покажчик-константа, дозволяється).

Арифметичні операції з покажчиками (складання з константою, віднімання, інкремент і декремент) автоматично враховують розмір типу величин, що адресуються покажчиками. Ці операції застосовні тільки до покажчиків одного типу і мають сенс в основному при роботі із структурами даних, послідовно розміщеними в пам'яті, наприклад, з масивами.

Інкремент переміщує покажчик до наступного елементу масиву, *декремент* – до попереднього. Фактично значення покажчика змінюється на величину sizeof (тип). Якщо покажчик на певний тип збільшується або зменшується на константу, його значення змінюється на величину цієї константи, помножену на розмір об'єкту даного типу, наприклад:

```
short * p = new short [5]:
 p++;            // значення p збільшується на 2
```

```
long * q = new long [5];
 q++;            // значення q збільшується на 4
```

Різниця двох покажчиків – це різниця їх значень, що ділиться на розмір типу в байтах (у застосуванні до масивів різниця покажчиків, наприклад, на третій і шостий елементи рівна 3). Підсумовування двох покажчиків не допускається.

При записі виразів з покажчиками слід звертати увагу на пріоритети операцій. Як приклад розглянемо послідовність дій, задану в операторі

```
*p++ = 10;
```

Операції розадресації та інкремента мають однаковий пріоритет і виконуються справа наліво, але, оскільки інкремент постфіксний, він виконується після виконання операції привласнення. Таким чином, спочатку за адресою, записаною в покажчику p, буде записане значення 10, а потім покажчик буде збільшений на кількість байт, відповідну його типу. Те ж саме можна записати докладніше:

```
*p = 10; p++;
```

Вираз (*p)++, навпаки, інкрементує значення, на яке посилається покажчик.

Унарна операція отримання адреси & застосовна до величин, що мають ім'я і розміщеним в оперативній пам'яті. Таким чином, не можна одержати адресу скалярного виразу, неіменованої константи або регістрової змінної. Приклади операції наводилися вище.

Ідентифікатор масиву є константним покажчиком на його нульовий елемент. Наприклад, для масиву з попереднього лістингу ім'я b – це те ж саме, що &b[0], а до i-му елемента масиву можна звернутися, використовуючи вираз *(b+i). Можна описати покажчик, привласнити йому адресу початку масиву і працювати з масивом через покажчик. Такий фрагмент програми копіює всі елементи масиву a в масив b:

```
Int a[100], b[100];
int *pa = a;
```

```
int *pb = b:
for (int i = 0; i<100; i++)
*pb++ = *pa++;      // або pb[i]= pa[i];
```

Динамічні масиви створюють за допомогою операції new, при цьому необхідно вказати тип і розмірність, наприклад:

```
int n = 100;
float *p = new float [n];
```

У цьому рядку створюється змінна-покажчик на float, в динамічній пам'яті відводиться безперервна область, достатня для розміщення 100 елементів дійсного типу, і адреса її початку записується в покажчик p. Динамічні масиви не можна при створенні ініціалізувати, і вони не обнуляються.

Перевага динамічних масивів полягає в тому, що розмірність може бути змінною, тобто об'єм пам'яті, що виділяється під масив, визначається на етапі виконання програми. Доступ до елементів динамічного масиву здійснюється точно так, як і до статичних, наприклад, до елемента номер 5 приведеного вище масиву можна звернутися як p[5] або *(p+5).

Альтернативний спосіб створення динамічного масиву – використання функції malloc бібліотеки C:

```
int n = 100:
float *q = (float *) malloc(n * sizeof(float));
```

Операція перетворення типу, записана перед зверненням до функції malloc, потрібна тому, що функція повертає значення покажчика типу void*, а ініціалізувався покажчик на float.

Пам'ять, зарезервована під динамічний масив за допомогою new [], повинна звільнятися операцією delete []. а пам'ять, виділена функцією malloc – за допомогою функції free, наприклад:

```
delete [] p; free (q);
```

При невідповідності способів виділення **і** звільнення пам'яті результат не визначений. Розмірність масиву в операції delete не вказується, але квадратні дужки обов'язкові.

Багатовимірні масиви задаються вказівкою кожного вимірювання в квадратних дужках, наприклад, оператор

```
int matr [5][6];
```

задає опис двовимірного масиву з 5 рядків і 6 стовпців. У пам'яті такий масив розташовується в послідовних осередках *відрядковий.* Багатовимірні масиви розміщуються так, що при переході до наступного елемента найшвидше змінюється останній індекс. Для *доступу* до елемента багатовимірного масиву вказуються всі його індекси, наприклад, matr[i][j], або більш екзотичним способом: *(matr[i]+j) або *(*(matr+i)+j). Це можливо, оскільки matr[i] є адресою початку і-го рядка масиву.

Для створення динамічного багатовимірного масиву необхідно вказати в операції new все його розмірності (найлівіша розмірність може бути змінною) наприклад:

```
int nstr = 5;
int ** m = (int **) new int [nstr][10];
```

Більш універсальний і безпечний спосіб виділення пам'яті під двовимірний масив, коли обидві його розмірності задаються на етапі виконання програми наведений далі:

```
int nstr, nstb;
cout << " Введіть кількість рядків і стовпців :";
cin >> nstr , nstb;
int **a = new int *[nstr];        // 1
for(int i = 0;  i<nstr;  i++)     // 2
a[i] = new int [nstb];            // 3
```

У операторі 1 оголошується змінна типу покажчик на покажчик на int і виділяється пам'ять під масив покажчиків на рядки масиву (кількість рядків nstr). У операторі 2 організовується цикл для виділення пам'яті під кожний рядок масиву. У операторі 3 кожному елементу масиву покажчиків на рядки привласнюється адреса початку ділянки пам'яті, виділеного під рядок двовимірного масиву. Кожен рядок складається з nstb елементів типу int.

Звільнення пам'яті з-під масиву з будь-якою кількістю вимірювань виконується за допомогою операції delete []. Покажчик на константу видалити не можна.

Завдання на лабораторну роботу

Скласти програму, що виконує з одновимірним масивом дії відповідно до варіанту завдання. Замість класичного доступу до елементів масиву (наприклад, **MyArray[i]**) і виконання операцій над елементами використовувати покажчики. Обраработку масиву виконати у функції. Функцію в головній програмі викликати через покажчик. Пам'ять під масив виділити динамічно.

Скласти програму, що виконує з одновимірним масивом такі дії:

Варіант 1

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- суму негативних елементів масиву;
- множення елементів масиву, розташованих між максимальним і мінімальним елементами.

Упорядкувати елементи масиву за збільшенням.

Варіант 2

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- суму позитивних елементів масиву;
- множення елементів масиву, розташованих між максимальним за модулем і мінімальним за модулем елементами.

Упорядкувати елементи масиву за спаданням.

Варіант 3

У одновимірному масиві, що складається з цілих елементів, обчислити:

- множення елементів масиву з парними номерами;
- суму елементів масиву, розташованих між першим і останнім нульовими елементами.

Перетворити масив так, щоб спочатку розташовувалися всі позитивні елементи, а потім – всі негативні (елементи, які дорівнюють 0, вважати позитивними).

Варіант 4

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- суму елементів масиву з непарними номерами;
- суму елементів масиву, розташованих між першим і останнім негативними елементами.

Стискувати масив, видаливши з нього всі елементи, модуль яких не перевищує 1. Елементи, що звільнилися в кінці масиву, заповнити нулями.

Варіант 5

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- максимальний елемент масиву;
- суму елементів масиву, розташованих до останнього позитивного елемента.

Стиснити масив, видаливши з нього всі елементи, модуль яких знаходиться в інтервалі [a, b]. Елементи, що звільнилися в кінці масиву, заповнити нулями.

Варіант 6

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- мінімальний елемент масиву;
- суму елементів масиву, розташованих між першим і останнім позитивними елементами.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, які дорівнюють нулю, а потім – всі інші.

Варіант 7

В одновимірному масиві, що складається з n цілих елементів, обчислити:

- номер максимального елементу масиву;
- множення елементів масиву, розташованих між першим і другим нульовими елементами.

Перетворити масив так, щоб в першій його половині розташовувалися елементи, що стояли в непарних позиціях, а в другій половині – елементи, що стояли в парних позиціях.

Варіант 8

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- номер мінімального елемента масиву;
- суму елементів масиву, розташованих між першим і другим негативними елементами.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, модуль яких не перевищує 1, а потім – всі інші.

Варіант 9

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- максимальний за модулем елемент масиву;
- суму елементів масиву, розташованих між першим і другим позитивними елементами.

Перетворити масив так, щоб елементи, які дорівнюють нулю, розташовувалися після всіх інших.

Варіант 10

В одновимірному масиві, що складається з n цілих елементів, обчислити:

- мінімальний за модулем елемент масиву;
- суму модулів елементів масиву, розташованих після першого елемента, який дорівнює нулю.

Перетворити масив так, щоб у першій його половині розташовувалися елементи, що стояли в парних позиціях, а в другій половині – елементи, що стояли в непарних позиціях.

Варіант 11

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- номер мінімального за модулем елемента масиву;
- суму модулів елементів масиву, розташованих після першого негативного елемента.

Стискувати масив, видаливши з нього всі елементи, величина яких знаходиться в інтервалі [a, b]. Елементи, що звільнилися в кінці масиву, заповнити нулями.

Варіант 12

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- номер максимального за модулем елемента масиву;
- суму елементів масиву, розташованих після першого позитивного елемента.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, ціла частина яких лежить в інтервалі [a, b], а потім – всі інші.

Варіант 13

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- кількість елементів масиву, розташованих в діапазоні від A до B;
- суму елементів масиву, розташованих після максимального елемента.

Упорядкувати елементи масиву за спаданням модулів елементів.

Варіант 14

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- кількість елементів масиву, які дорівнюють 0;
- суму елементів масиву, розташованих після мінімального елемента.

Упорядкувати елементи масиву за збільшенням модулів елементів.

Варіант 15

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- кількість елементів масиву, які більше 3;
- множення елементів масиву, розташованих після максимального за модулем елемента.

Перетворити масив так, щоб спочатку розташовувалися всі негативні елементи, а потім – всі позитивні (елементи, які дорівнюють 0, вважати позитивними).

Варіант 16

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- кількість негативних елементів масиву;
- суму модулів елементів масиву, розташованих після мінімального за модулем елемента.

Замінити всі негативні елементи масиву їх квадратами і упорядкувати елементи масиву за збільшенням.

Варіант 17

В одновимірному масиві, що складається з n цілих елементів, обчислити:

- кількість позитивних елементів масиву;
- суму елементів масиву, розташованих після останнього елемента, який дорівнює нулю.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, ціла частина яких не перевищує 1, а потім – всі інші.

Варіант 18

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- кількість елементів масиву, які менше 3;
- суму цілих частин елементів масиву, розташованих після останнього негативного елемента.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, що відрізняються від максимального не більше ніж на 20 %, а потім – всі інші.

Варіант 19

У одновимірному масиві, що складається з n дійсних елементів, обчислити:

- множення негативних елементів масиву;
- суму позитивних елементів масиву, розташованих до максимального елемента.

Змінити порядок проходження елементів у масиві на зворотний.

Варіант 20

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

- множення позитивних елементів масиву;
- суму елементів масиву, розташованих до мінімального елемента.

Упорядкувати за збільшенням окремо елементи, що стоять на парних місцях, і елементи, що стоять на непарних місцях.

Контрольні запитання

1. Дайте визначення динамічного масиву.
2. Які операції можливі з покажчиками?
3. Вкажіть приклади завдання розмірності динамічного масиву.
4. Як можна ініціалізувати елементи массиву з допомогою покажчиків?
5. Яке інформаційне навантаження несе ім'я масиву?
6. Яка операція використовується для визначення адреси довільного елемента масиву?

Лабораторна робота № 8

Підготовка і розв’язання на ПЕОМ задач з використанням рядків і файлів

Мета лабораторної роботи – знайомство з можливостями введення-виведення даних, освоєння методики обробки інформації з використанням рядків, а також препроцесорної обробки даних, освоєння методики обробки інформації з використанням багатофайлових програм.

Перед виконанням лабораторної роботи студент повинен знати:

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

Після виконання лабораторної роботи студент повинен вміти:

- виконувати основні операції з рядками: створення, копіювання, конкатенацію, пошук підрядка в рядку і т. д.;
- розробляти програми для вирішення завдань з використанням багатофайлових програм.

Теоретичний матеріал

Рядки

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* :

\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.

Зауваження. Значення типу char, що 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. У числі символів рядка можуть знаходитися будь-які символьні константи, наприклад

Дзвінок в кінці повідомлення \007\n.

Сусідні рядкові константи транслятором "склеюються". Наприклад

АБВ ДЕ означає те ж, що АБВГДЕ.

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

char s[] = АБВГДЕ;

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

Рішення

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

Програму можна зробити трохи коротше, переписавши оператор циклу:

for (i = 0; s2[i]=s1[i]; i++);

Якщо пригадати про покажчики на символи, можна написати і так:

for (char *p1 = s1, *p2 = s2; *p2++=*p1++;);

Зауваження. Кома в C++ є не тільки роздільником, але і оператором послідовного виконання. Значенням цієї операції є значення найправішого операнда.

Рядкові бібліотечні функції

Функції для роботи з рядками описані в заголовному файлі string.h. 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 в новостворюваний функцією malloc() область пам'яті. Повертає покажчик на створену копію або 0 при невдачі. Програміст відповідає за звільнення пам'яті.

Приклад

```
char *dup_str = strdup (string);
...
    free (dup_str);
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 всіх 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.

Зауваження. Для кожної функції існує варіант з дальніми покажчиками. Його ім'я, як правило, має префікс "_f".

Різновиди введення і виведення

Засоби введення і висновку формально не входять в стандарт мов C і C++, але фактично стандартизовані і містяться в системних бібліотеках функцій.

У них можна виділити такі групи:

1) консольні – орієнтовані на введення з клавіатури і виведення на дисплей. Описані в заголовному файлі conio.h.

2) файлові – призначені для роботи з файлами. Описані в io.h.

3) потокові – аналогічні файловим, але надають більший сервіс програмісту. Описані в stdio.h.

4) засоби ДОС – введення і виведення функціями операційної системи. Описані в dos.h.

5) об'єктні – об'єктно-орієнтоване введення/виведення, тільки в C++. Описані в iostream.h, fstream.h, iomanip.h.

Відкриття і закриття потоку

Схема роботи з потоком така ж, як і з файлом: відкрити потік, виконати читання і/або запис, закрити потік.

Відкриває потік функція

```
FILE* fopen(
const char *filename,// ім'я файлу, що асоціюється з потоком
const char *mode     // рядок режимів роботи з потоком
```

) – повертає покажчик на потік, який ідентифікує його в подальших операціях.

У рядку режимів можуть знаходитися такі символи:

r – відкрити тільки для читання.

w – створити для запису. Існуючий файл буде перекритий новим.

a – відкрити для дозапису, або створити для запису, якщо файл не існує.

+ – операції виконуватимуться з вже існуючим файлом.

t – текстовий режим (обробка символів CR-LF).

b – двійковий режим (ніякої обробки).

За відсутності в рядку b або t, режим визначається глобальній змінній _fmode, визначеній в заголовному файлі fcntl.h.

FILE – це структура для потоку, що управляє, оголошена в stdio.h. Вона не призначена для прямого використання.

Потік є програмною надбудовою над файлом, що надає програмісту додатковий сервіс. Крім сумісного відкриття потоку і файлу (fopen), можна відкрити потік і асоціювати його з вже відкритим файлом (fdopen), відкрити файл і асоціювати його з вже відкритим потоком (freopen).

Закриває потік і вивантажує буфери функція

int fclose(FILE *stream) – повертає 0 при успіху і EOF при помилці.

EOF - константа, визначена в stdio.h.

int *fcloseall(*void*)* – закриває всі відкриті потоки, окрім стандартних: stdin, stdout, stdprn, stderr і stdaux.

Введення і виведення символів

Читання символу з потоку виконується функцією

int *fgetc(*FILE **stream)* – повертає код символу. При помилці повертає EOF.

Запис символу в потік виконується функцією

int *fputc(*int *з,* FILE **stream*) – повертає код символу з. При помилці повертає EOF.

Читання символу із *стандартного* потоку stdin виконується функцією int *fgetchar(void);*

Запис символу в стандартний потік stdout виконується функцією int *fputchar(int з);*

Завдання. Скопіювати файл xxx.bin у файл yyy.bin.

Рішення

```
#include <stdio.h>
void main()
{
  FILE *in  = fopen ("xxx.bin","r");
  FILE *out = fopen ("yyy.bin","w");
  char ch;
  if (!in) return;
  while (1) {
    if ((ch=fgetc(in))==EOF) break;
    fputc(ch, out);
  }
  fcloseall();
}
```

Введення і виведення рядків

Читання рядка з потоку виконується функцією

char *fgets(

char *s, // покажчик на буфер, що приймає рядок

int n, // гранична кількість читаних символів (звичайно розмір буфера)

FILE *stream

) – повертає покажчик на буфер або NULL при помилці.

Читання припиняється, коли досягнутий кінець рядка або прочитано n-1 символів з файлу. Рядок в буфері замикається нульовим символом. Покажчик файлу переміщається за символи CR-LF.

Запис рядка в потік виконується функцією

char *fputs(

char *s, // покажчик на рядок*

FILE *stream*

) – повертає покажчик на останній записаний символ або EOF при помилці.

Термінальний символ рядка не копіюється. Символи CR-LF додаються у файл.

Завдання. Скопіювати текстовий файл xxx.txt у файл yyy.txt по рядках.

Рішення (один з можливих варіантів)

```
#include <stdio.h>
void main() {
  const int n = 100;
  char buf [n];
  FILE *in  = fopen ("xxx.txt","rt");
  FILE *out = fopen ("yyy.txt","wt");
  if (!in) return;
  while (!feof(in)) {
     if(fgets(buf, n, in)==NULL) continue;
     fputs(buf, out);
  }
  fcloseall();
}
```

Введення і виведення записів

Читання записів з потоку виконується функцією

size_t fread(

void *ptr, // покажчик на буфер в пам'яті, що приймає записи

size_t size, // розмір запису в байтах

size_t n, // кількість читаних записів

FILE *stream

) – при успіху повертає n, при невдачі – кількість прочитаних записів, можливо нуль.

Загальна кількість читаних байтів рівне n *size.

Виведення записів в потік виконується функцією

size_t fwrite(

const void *ptr,

size_t size

size_t n

FILE* stream

) – додає вказану кількість записів у файл.

Сенс параметрів і значення, що повертається, той же, що у функції fread.

Управління покажчиком файлу

Читання і запис виконуються в тому місці файлу, де знаходиться покажчик файлу. Встановити покажчик можна функцією

int fseek(FILE *stream,

long offset, // зсув покажчика

int whence // відлік зсуву

) – повертає 0 при успіху.

При помилці, викликаною неможливістю відкрити файл або пристрій, повертає ненульове значення. Інші помилки не діагностуються.

Для вказівки точки відліку зсуву використовують константи:

SEEK_SET = 0 – відлік від початку файлу;

SEEK_CUR = 1 – відлік від кінця файлу;

SEEK_END = 2 – відлік від поточної позиції покажчика.

Функція

long ftell(FILE *stream) – повертає поточну позицію покажчика.

При помилці повертає 1 і встановлює глобальну змінну errno в ненульове значення.

Зауваження. Ті ж дії виконуються функціями fsetpos і fgetpos.

Стан потоку

Макрос, перевіряючий досягнення *кінця файлу* потоку:

int feof (FILE *stream) – повертає не 0, якщо досягнутий кінець файлу і 0 - інакше.

Макрос, що тестує індикатор помилки потоку:

int ferror(FILE *stream) – повертає не 0, якщо виявлена помилка запису або читання.

Одного разу встановлений індикатор помилки зберігається до виконання функцій clearerr, rewind або закриття потоку. Індикатор "кінець файлу" встановлюється заново кожною операцією читання.

void clearerr(FILE *stream) – обнуляє індикатори помилки і кінця файлу.

void rewind(FILE *stream) – робить те ж, що clearerr, а також встановлює покажчик в початок файлу.

При виникненні помилки глобальна змінна errno (визначена у файлах errno.h, stddef.h, stdlib.h) одержує ненульовий номер помилки.

Форматоване виведення

Розглянуті функції виводять інформацію в потік без або майже без перетворення. Функція fprintf перетворить дані, що виводяться, в послідовність символів, керуючись рядком формату.

int fprintf (FILE *stream,

const char *format // рядок формату

[, argument, ...] // значення, що виводяться

) – при успіху повертає кількість виведених байт, інакше – EOF.

Квадратні дужки говорять про необов'язковість аргументу

Рядок формату містить прості символи і специфікації формату. Прості символи копіюються у вихідний потік без зміни, специфікації застосовуються для форматування решти аргументів функції. Якщо аргументів менше, ніж специфікацій, наслідки непередбачувані. Якщо аргументів більше, ніж специфікацій, зайві аргументи ігноруються.

Загальний вид специфікації формату такий:

%[прапори] [ширина] [.точність] [розмір] тип

прапори – ознаки вирівнювання, використання знаків, десяткової крапки, кінцевих нулів, 8-ічних і 16-ічних префіксів;

ширина – мінімальне число друкованих символів з урахуванням пропусків і нулів;

точність – максимальне число друкованих символів (для цілих – мінімальне число цифр);

розмір – визначає розмір аргументу;

тип – символ специфікації типу – обов'язковий елемент формату.

Форматоване введення

Для форматованого введення з потоку застосовують функцію

int fscanf (FILE *stream,

const char *format

[, address, ...]

) – повертає число полів введення тих, що відформатували і розміщених в пам'яті. При невдачі повертає EOF.

Функція fscanf розглядає вхідний потік як послідовність полів введення. Поле введення закінчується:

першим символом пропуску (але не включає його);

першим символом, який не може бути перетворений за специфікацією формату, зіставленому цьому полю;

(n+1)-м символом, якщо специфікація включає ширину поля в n символів.

Функція проглядає послідовність полів введення, форматує їх і розміщує за адресами – аргументам fscanf. Число адрес, специфікацій формату і полів введення повинно бути узгоджено.

Рядок формату складається з символів (' ',\t,\n), що невідображаються, символів (всі інші, окрім '%'), що відображаються, і специфікаторів формату.

Якщо fscanf зустрічає символ, що невідображається, в рядку, формату, вона прочитуватиме, але не зберігати всі символи вхідного потоку, що невідображаються, аж до першого символу, що відображається.

Якщо fscanf зустрічає символ, що відображається, в рядку, формату, вона прочитає, але не збереже відповідний символ вхідного потоку.

Специфікація формату наказує fscanf читання, перетворення і розміщення в пам'яті одного вхідного поля.

Загальний вид специфікації формату:

%[*] [ширина] [модіф. розміру] [модіф. типу арг.] символ типу

* – відміняє привласнення поля введення;

ширина – максимальне число прочитуваних символів;

модифікатор розміру – N - near, F - far;

модифікатор типу аргументу – змінює тип адресного аргументу;

символ типу – символ специфікації типу – обов'язковий елемент формату.

У табл. 9 перераховані інші функції виведення, формату, з вказівкою заголовного файлу і вихідного потоку.

Таблиця 9

Функції виведення з вказівкою заголовного файлу і вхідного потоку

cprintf	CONIO.H	Консоль
fprintf	STDIO.H	Потік
printf	STDIO.H	stdout
sprintf	STDIO.H	Рядок

У табл. 10. перераховані інші функції введення, формату, з вказівкою заголовного файлу і вхідного потоку.

Таблиця 10

Функції виведення з вказівкою заголовного файлу і вхідного потоку

cscanf	CONIO.H	Консоль
fscanf	STDIO.H	Потік
scanf	STDIO.H	stdin
sscanf	STDIO.H	Рядок

Завдання на лабораторну роботу

Виконати вправи поточної лабораторної роботи, оформивши програмний продукт у вигляді багатофайлового проекту.

Усі необхідні дані для функцій повинні передаватися їм як параметри. Використання глобальних змінних у функціях не допускається.

За допомогою текстового редактора створити файл, що містить текст, довжина якого не перевищує 1 000 символів (довжина рядка не повинна перевищувати 70 символів). Ім'я файлу повинне мати розширення DAT.

Варіант 1

Написати програму, яка:

- виводить текст на екран дисплея;
- по натисненню довільної клавіші по черзі виділяє кожне речення тексту;
- визначає кількість речень в тексті.

Варіант 2

Написати програму, яка:

- виводить текст на екран дисплея;
- по натисненню довільної клавіші по черзі виділяє кожне слово тексту;
- визначає кількість слів в тексті.

Варіант 3

Написати програму, яка:

- виводить текст на екран дисплея;
- по натисненню довільної клавіші по черзі виділяє кожне слово тексту, що закінчується на голосну букву;
- визначає кількість слів в тексті, що закінчуються на голосну букву.

Варіант 4

Написати програму, яка:

- виводить текст на екран дисплея;
- по натисненню довільної клавіші по черзі виділяє кожне речення тексту в послідовності 2, 3, 1.

Варіант 5

Написати програму, яка:

- виводить текст на екран дисплея;
- по натисненню довільної клавіші по черзі виділяє кожне із слів тексту, у яких перший і останній символи співпадають;
- визначає кількість слів тексту, у яких перший і останній символи співпадають.

Варіант 6

Написати програму, яка:

- виводить текст на екран дисплея;
- по натисненню довільної клавіші по черзі виділяє кожне слово тексту, що починається на голосну букву;
- визначає кількість слів в тексті, що починаються на голосну букву.

Варіант 7

Написати програму, яка:

- виводить текст на екран дисплея;
- визначає кількість символів в щонайдовшому слові;
- по натисненню довільної клавіші по черзі виділяє кожне слово тексту, що містить максимальну кількість символів.

Варіант 8

Написати програму, яка:

- виводить текст на екран дисплея;
- визначає кількість символів в найкоротшому слові;

• по натисненню довільної клавіші по черзі виділяє кожне слово тексту, що містить мінімальну кількість символів.

Варіант 9

Написати програму, яка:

• виводить текст на екран дисплея;

• визначає в кожнім реченні тексту кількість символів, відмінних від букв і пропуску;

• по натисненню довільної клавіші по черзі виділяє кожне речення тексту, а у виділеній пропозиції – по черзі всі символи, відмінні від букв і пропуску.

Варіант 10

Написати програму, яка:

• виводить текст на екран дисплея;

• визначає кількість пропозицій тексту і кількість слів в кожнім реченні;

• по натисненню довільної клавіші по черзі виділяє кожне речення тексту, а у виділенім реченні – по черзі всі слова.

Варіант 11

Написати програму, яка:

• виводить текст на екран дисплея;

• визначає кількість букв ‘а’ в останньому слові тексту;

• по натисненню довільної клавіші по черзі виділяє останнє слово, а у виділеному слові по черзі всі букви ‘а’.

Варіант 12

Написати програму, яка:

• виводить текст на екран дисплея;

• визначає щонайдовшу послідовність цифр в тексті (вважати, що будь-яка кількість пропусків між двома цифрами не перериває послідовності цифр);

• по натисненню довільної клавіші по черзі виділяє кожну послідовність цифр, що містить максимальну кількість символів.

Варіант 13

Написати програму, яка:

• виводить текст на екран дисплея;

• визначає порядковий номер заданого слова в кожнім реченні тексту;

• по натисненню довільної клавіші по черзі виділяє кожне речення тексту, а у виділеній реченні – задане слово.

Варіант 14

Написати програму, яка:

• виводить текст на екран дисплея;

• по натисненню довільної клавіші по черзі виділяє в тексті задане слово (задане слово вводиться з клавіатури);

• виводить текст на екран дисплея ще раз, викидаючи з нього задане слово і видаляючи зайві пропуски.

Варіант 15

Написати програму, яка:

• виводить текст на екран дисплея;

• по натисненню довільної клавіші по черзі виділяє в тексті задані слова, які потрібно поміняти місцями (задані слова вводиться з клавіатури);

• виводить текст на екран дисплея ще раз, міняючи в ньому місцями задані слова і видаляючи зайві пропуски.

Варіант 16

Написати програму, яка:

• виводить текст на екран дисплея;

• по натисненню довільної клавіші по черзі виділяє в тексті задане слово (задане слово вводиться з клавіатури);

• виводить текст на екран дисплея ще раз, беручи задане слово в лапки, і по черзі виділяє задане слово разом з лапками.

Варіант 17

Написати програму, яка:

• виводить текст на екран дисплея;

• виводить текст на екран дисплея ще раз, вставляючи в кожну пропозицію як остання задане слово (задане слово вводиться з клавіатури);

• по натисненню довільної клавіші по черзі виділяє в тексті вставле-не слово.

Варіант 18

Написати програму, яка:

- виводить текст на екран дисплея;
- по натисненню довільної клавіші по черзі виділяє в тексті зайві пропуски між словами;
- виводить текст на екран дисплея ще раз, прибираючи зайві пропуски між словами і починаючи кожну пропозицію з нового рядка.

Варіант 19

Написати програму, яка:

- виводить текст на екран дисплея;
- по натисненню довільної клавіші по черзі виділяє в тексті задане слово (задане слово вводиться з клавіатури);
- виводить текст на екран дисплея ще раз, замінюючи в заданому слові рядкові букви прописними.

Варіант 20

Написати програму, яка:

- виводить текст на екран дисплея;
- визначає найбільшу кількість підряд пропусків, що йдуть, в тексті;
- по натисненню довільної клавіші по черзі виділяє кожну з послідовностей пропусків максимальної довжини.

Контрольні питання

1. На які групи можна розділити бібліотечні функції введення-висновку?
2. Що таке потік?
3. Чи можна відкрити потік, не відкриваючи файлу?
4. Як відкрити потік в двійковому режимі?
5. Що повертає функція fopen?
6. Яка функція виводить символ в стандартний вивідний потік?
7. Які функції читають і записують рядок в потік?
8. Як встановити покажчик на кінець потоку, відкритого для читання?
9. Як перевірити, чи досяг покажчик потоку кінця файлу?

10. Як перевірити, чи немає помилки при роботі з потоком?
11. Чи можна скинути індикатор помилки, не закриваючи потоку?
12. Яка функція виконує форматований висновок в потік?
13. Чим відрізняється функція printf від функції fprintf?
14. Що таке поле введення для функції fscanf?
15. Який влаштований рядок формату функції fscanf?

Лабораторна робота № 9

Підготовка і розв'язання на ПЕОМ задач з використанням рядків і макросів

Мета лабораторної роботи – знайомство з можливостями препроцесорної обробки даних, освоєння методики обробки інформації з використанням багатофайлових програм.

Перед виконанням лабораторної роботи студент повинен знати:

визначення і використання засобів препроцесорної обробки;

основні засоби роботи з макросами.

Після виконання лабораторної роботи студент повинен вміти:

розробляти програми для вирішення завдань з використанням багатофайлових проектів.

Теоретичний матеріал

Стадії і команди препроцесорної обробки

У інтегроване середовище підготовки програм на C++ в компілятор мови, як обов'язковий компонент, входить препроцесор. Призначення препроцесора – обробка початкового тексту програми до її компіляції.

Препроцесорна обробка відповідно до вимог стандарту мови C++ включає декілька стадій, що виконуються послідовно. Конкретна реалізація транслятора може об'єднувати декілька стадій, але результат повинен бути таким, неначебто вони виконувалися послідовно:

всі системно-залежні позначення (наприклад, системно залежний індикатор кінця рядка) перекодують в стандартні коди;

кожна пара з символів \' і "кінець рядка" забираються, і тим самим наступний рядок початкового файлу приєднується до рядка, в якому знаходилася ця пара символів;

у тексті розпізнаються директиви препроцесора, а кожен коментар замінюється одним символом порожнього проміжку;

виконуються директиви препроцесора і проводяться макропідстановки;

ESC-послідовності в символьних константах і символьних рядках, наприклад, '\ п' замінюються на їх еквіваленти (на відповідні числові коди);

суміжні символьні рядки конкатенуються, тобто з'єднуються в один рядок.

Знайомство з перерахованими завданнями препроцесорної обробки пояснює деякі угоди синтаксису мови. Наприклад, стає зрозумілим сенс тверджень: кожен символьний рядок може бути перенесений у файлі на наступний рядок, якщо використовувати символ \' або "два символьні рядки, записані поряд, сприймається як один рядок.

Розглянемо детально стадію обробки директив препроцесора. При її виконанні можливі такі дії:

заміна ідентифікаторів (позначень) наперед підготовленими послідовностями символів;

включення в програму текстів з вказаних файлів;

виключення з програми окремих частин її тексту (умовна компіляція);

макропідстановка, тобто заміна позначення текстом, що параметризується, формованим препроцесором з урахуванням конкретних параметрів (аргументів).

Для управління препроцесором, тобто для завдання потрібних дій, використовуються команди (директиви) препроцесора, кожна з яких поміщається на окремому рядку і починається з символу #. Визначені такі препроцесорні директиви: #define, #include, #undef, #if, #ifdef, #ifndef, #else, #endif, #elif, #line. #error, #pragma #.

Директива #define має декілька модифікацій. Вони передбачають визначення макросів або препроцесорних ідентифікаторів, кожному з яких ставиться у відповідність деяка символьна послідовність. У подальшому тексті програми препроцесорні ідентифікатори замінюються на наперед заплановані послідовності символів.

Директива #include дозволяє включати в текст програми текст з вибраного файлу.

Директива #undef відміняє дію команди #define, яка визначила до цього ім'я препроцесорного ідентифікатора.

Директива #if і її модифікації #ifdef, #ifndef спільно з директивами #else, #endif, #elif дозволяють організувати умовну обробку тексту програми. Умовність полягає в тому, що компілюється не весь текст, а тільки ті його частини, які так чи інакше виділені за допомогою перерахованих директив.

Директива #line дозволяє управляти нумерацією рядків у файлі з програмою. Ім'я файлу і початковий номер рядка указуються безпосередньо в директиві #line.

Директива #error дозволяє задати текст діагностичного повідомлення, яке виводиться при виникненні помилок.

Директива #pragma викликає дії, залежні від реалізації.

Директива # нічого не викликає, оскільки є порожньою директивою, тобто не дає ніякого ефекту і завжди ігнорується.

Включення текстів з файлів

Для включення тексту з файлу використовується команда #include, що має дві форми запису:

#include <ім'я_файла> // Ім'я в кутових дужках

#include "ім'я_файла" // Ім'я в лапках

Якщо ім'я_файла – в кутових дужках, то препроцесор розшукує файл в стандартних системних каталогах. Якщо ім'я_файла поміщене в лапки, то спочатку препроцесор проглядає поточний каталог користувача і тільки тоді звертається до проглядання стандартних системних каталогів.

Починаючи працювати з мовою C++, користувач відразу ж стикається з необхідністю використання в програмах засобів введення-висновку. Для цього на початку тексту програми поміщають директиву:

#include <iostream.h>

Виконуючи цю директиву, препроцесор включає в програму засобу зв'язку з бібліотекою введення-висновку. Пошук файлу iostream.h ведеться в стандартних системних каталогах.

За прийнятою угодою суфікс .h приписується тим файлам, які потрібно поміщати в заголовку програми, тобто до виконуваних операторів.

Окрім такого деякою мірою стандартного файлу, яким є iostream.h, у заголовок програми можуть бути включені будь-які інші файли (стандартні або підготовлені спеціально).

Заголовні файли виявляються вельми ефективним засобом при модульній розробці крупних програм, коли зв'язок між модулями, що розміщуються в різних файлах, реалізується не тільки за допомогою параметрів, але і через зовнішні об'єкти, глобальні для декількох або всіх модулів. Описи таких зовнішніх об'єктів (змінних, масивів, структур і т. п.) поміщаються в одному файлі, який за допомогою директив #include включається у всі модулі, де необхідні зовнішні об'єкти. У той же файл можна включити і директиву підключення бібліотеки функцій введення-виведення.

У заголовному файлі прийнято розміщувати:

визначення типів, що задаються користувачем, констант, шаблонів;

оголошення (прототипи) функцій;

оголошення зовнішніх глобальних змінних (з модифікатором extern);

простори імен.

Заголовний файл може бути, наприклад, таким:

```
#include <iostream>    // Включення засобів обміну
extern int ii, jj, 11;         // Цілі зовнішні змінні
extern float AA, BB;      // Дійсні зовнішні змінні
```

У практиці програмування на C++ звичайна і в деякому розумінні зворотна ситуація. Якщо в програмі використовується декілька функцій, то іноді зручно текст кожній з них зберігати в окремому файлі. При підготовці програми користувач включає в неї тексти використовуваних функцій за допомогою команд #include.

Умовна компіляція

Умовна компіляція забезпечується в мові C++ набором команд, які, по суті, управляють не компіляцією, а препроцесорною обробкою:

```
#if константний вираз
#ifdef  ідентифікатор
#ifndef  ідентифікатор
```

#else

#endif

#elif

Перші три команди виконують перевірку умов, дві наступні – дозволяють визначити діапазон дії умови, що перевіряється. Команду #elif розглянемо дещо пізніше. Загальна структура застосування директив умовної компіляції така:

#if ... текст 1

#else текст_2

#endif

Конструкція #else текст_2 не обов'язкова. Текст_1 включається в компільований текст тільки при істинності умови, що перевіряється. Якщо умова помилкова, то за наявності директиви #else на компіляцію передається текст_2. Якщо директива #else відсутня, то весь текст від #if до #endif за помилкової умови опускається. Відмінність між формами команд #if полягає в наступному.

У першій з перерахованих директив #if перевіряється значення константного цілочисельного виразу. Якщо він відмінний від нуля, то вважається, що умова, що перевіряється, істинна. Наприклад, в результаті виконання директив:

#if 5+4

текст_1

#endif

текст_1 завжди буде включений в компільовану програму.

У директиві #ifdef перевіряється, чи визначений за допомогою команди #define до теперішнього моменту ідентифікатор, поміщений після #ifdef. Якщо ідентифікатор визначений як препроцесорний, то текст_1 використовується компілятором.

У директиві #ifndef перевіряється зворотна умова – істинною вважається невизначеність ідентифікатора, тобто той випадок, коли ідентифікатор не був використаний в команді #define або його визначення було відмінено командою #undef.

Умовну компіляцію зручно застосовувати при відладці програм для включення або виключення контрольного друку. Наприклад

```
#define DE 1
#ifdef DE
cout << "Налагоджувальний друк";
#endif
```

Такого друку, що з'являється в програмі залежно від визначеності ідентифікатора DE, може бути декілька і, прибравши директиву #define DE 1, відразу ж відключаємо весь налагоджувальний друк.

Файли, призначені для препроцесорного включення в модулі програми; звичайно забезпечують захистом від повторного включення. Таке повторне включення може відбутися, якщо декілька модулів, в кожному з яких заплановано препроцесорне включення одного і того ж файлу, об'єднуються в загальний текст програми. Наприклад, такими засобами захисту забезпечені всі заголовні файли (подібні iostream.h) стандартної бібліотеки. Схема захисту від повторного включення може бути такою:

```
// Файл з іменем filename
 #ifndef _FILE_NAME
// Текст файлу filename, що включається
#define _FILE_NAME 1
endif
```

Тут _FILE_NAME – зарезервований для файлу filename препроцесорний ідентифікатор, який не повинен зустрічатися в інших текстах програми.

Для організації розгалуджень під час обробки препроцесором початкового тексту програми введена директива

#elif константний вираз

Структура початкового тексту із застосуванням цієї директиви така:

```
#if
текст_для_ if
#elif вираз_1
текст_1
```

#elif вираз_2
текст_2
#else
текст_для_випадку_else
#endif

Препроцесор перевіряє спочатку умову в директиві #if, якщо вона помилкова (дорівнює 0) – обчислює вираз_1, якщо вираз_1 рівний 0 – обчислюється вираз_2 і т. д. Якщо всі вирази помилкові, то в компільований текст включається текст_для_випадку_else. Інакше, тобто при появі хоч би одного дійсного виразу (у #if або в #elif), починає оброблятися текст, розташований безпосередньо за цією директивою, а вся решта директив не розглядається. Таким чином, препроцесор обробляє завжди тільки одну з ділянок тексту, виділених командами умовної компіляції.

Макропідстановки засобами препроцесора

Макрос, за визначенням, є засіб заміни однієї послідовності символів іншою. Для виконання замін повинні бути задані відповідні макроозначення. Просте макроозначення ми вже ввели, розглядаючи директиву

#define ідентифікатор рядок заміщення

Така директива зручна, проте вона має істотний недолік – рядок заміщення фіксоване. Великими можливостями володіє наступне макроозначення з параметрами

#define ім'я (список_параметрів) рядок__заміщення

Тут ім'я – ім'я макросу (ідентифікатор), список_параметрів – список розділених комами ідентифікаторів. Між ім'ям макросу і списком параметрів не повинно бути пропусків.

Класичний приклад макроозначення:

#define max(a,b) (a < b ? b : a)

дозволяє формувати в програмі вираз, визначальний максимальний з двох значень аргументів. При такому визначенні входження в програму

max(X,Y) замінюється виразом

(X < Y ? Y : X)

а використання

max(Z,4) приведе до формування виразу

(Z < 4 ? 4 : Z)

У першому випадку при дійсному значенні x < y повертається значення Y, інакше – значення x. У другому прикладі z порівнюється з константою 4 і вибирається більше із значень.

Не менш часто використовується визначення

#define ABS(X) (X < 0 ? - (X) : X)

З його допомогою в програму можна вставляти вираз для визначення абсолютних значень змінних. Конструкція

ABS(E - Z)

замінюється виразом

(E - Z < 0 ? -(E - Z) : E-Z)

в результаті обчислення якого визначається абсолютне значення виразу E - Z.

Порівнюючи макроси з функціями, найчастіше зазначають, що на відміну від функції, визначення якої завжди присутнє в одному екземплярі, коди, що формуються макросом, вставляються в програму стільки разів, скільки разів використовується макрос. У цьому відношенні макроси подібні вбудовуваним (online) функціям, але на відміну від вбудовуваних функцій підстановка для макросу виконується завжди. Звернемо увагу на ще одну відмінність: функція визначена для даних того типу, який вказаний в специфікації її параметрів і повертає значення тільки одного конкретного типу. Макрос придатний для обробки параметрів будь-якого типу, допустимих у виразах, що формуються при обробці рядка заміщення. Тип набуваючого значення залежить тільки від типів параметрів і від самих виразів. Таким чином, макрос може замінювати декілька функцій. Наприклад, приведені макроси max () і ABS() правильно працюють для параметрів з цілими або плаваючими типами, а результат залежить тільки від типів параметрів. Механізм перевантаження і шаблони функцій дозволяють вирішувати ті ж задачі, що і макроси. Саме тому на відміну від мови C в програмах на C++ макрозасоби використовуються рідше.

Завдання на лабораторну роботу

Виконати вправи поточної лабораторної роботи, оформивши програмний продукт у вигляді багатофайлового проекту.

Усі необхідні дані для функцій повинні передаватися їм як параметри. Використання глобальних змінних в функціях забороняється.

При обробці даних використати користувальницькі макроси з параметрами.

Варіанти для виконання завдання знаходяться в лабораторній роботі № 8 методичних рекомендацій.

Контрольні питання

1. Дайте визначення макросу.
2. Для чого використовується умовна компіляція?
3. Як можно захиститись від повторного включення модулів в заголовних файлах?
4. Які дії можно виконувати з допомогою директив препроцесора?
5. Що розміщується в заголовних файлах?

Рекомендована література

1. Вирт Н. Алгоритмы и структуры данных / Н. Вирт ; пер. с англ. – М. : Мир, 1989. – 360 с.

2. Глушаков С. В. Язык программирования С++ / С. В. Глушаков, А. В. Коваль, С. В. Смирнов. – Х. : Фолио, 2001. – 500 с.

3. Жарков В. Visual C++ 2008 в учебе, науке и технике / В. Жарков. – М. : Жарков Пресс, 2009. – 814 с.

4. Керниган Б. Язык программирования Си. Задачи по языку Си / Б. Керниган, Д. Ритчи, А. Фьюер. – М. : Финансы и статистика, 1985. – 279 с.

5. Павловская Т. А. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. – СПб. : Питер, 2006. – 461 с.

6. Подбельский В. В. Программирование на языке Си / В. В. Подбельский, С. С. Фомин. – М. : Финансы и статистика, 2004. – 600 с.

7. Подбельский В. В. Язык С++ : учебн. пособ. / В. В. Подбельский – 4-е изд. – М. : Финансы и статистика, 1999. – 560 с.

8. Пирогов В. Ю. Программирование на Visual C++.NET / В. Ю. Пирогов. – СПб. : БХВ-Петербург, 2003. – 800 с.

9. Тарасов О. В. Методичні рекомендації до виконання практичних завдань з навчальної дисципліни "Основи програмування та алгоритмічні мови" / О. В. Тарасов, В. М. Федорченко, М.Ю. Лосєв. – Х. : ХНЕУ, 2010. – 90 с.

10. Хортон А. Visual C++ 2005 : базовый курс / А. Хортон ; пер. с англ. – М. : Вильямс, 2007. – 1152 с.

11. Шилдт Г. Самоучитель С++ / Шилдт Г. ; пер. с англ. – СПб. : BHV-Санкт-Петербург, 1998. – 620 с.

Додатки

ДОДАТОК А

Приклад оформлення звіту з лабораторної роботи

Міністерство освіти і науки, молоді та спорту України
Харківський національний економічний університет
Кафедра інформаційних систем

Звіт
з лабораторної роботи № 1
дисципліни "Алгоритмізація та програмування"
на тему : "Вирішення на комп'ютері завдань лінійного характеру"

Виконав:
студент 1 курсу 1 групи
факультету ЕІ
Петров
Валерій Борисович

Перевірив:
доцент кафедри ІС
Іванов О. В.

Харків, 2012

Завдання № 1 Варіант № 5

Завдання: Розробити програму для знаходження модуля суми двох цілих чисел.

Специфікація програмних вимог

Формулювання

Задано два довільні цілі числа. Обчислити модуль суми цих чисел і вивести її на екран.

Вхідні дані

Два цілі числа, кожне за модулем не перевищує 109.
Прочитуються з клавіатури. Завжди коректні.

Обробка

Введені числа складаються та обчислюється модуль отриманої суми.

Вихідні дані

Ціле число, виводиться на екран.

Інтерфейс користувача
Введіть перше число
10
Введіть друге число
-20
Сума дорівнює 10

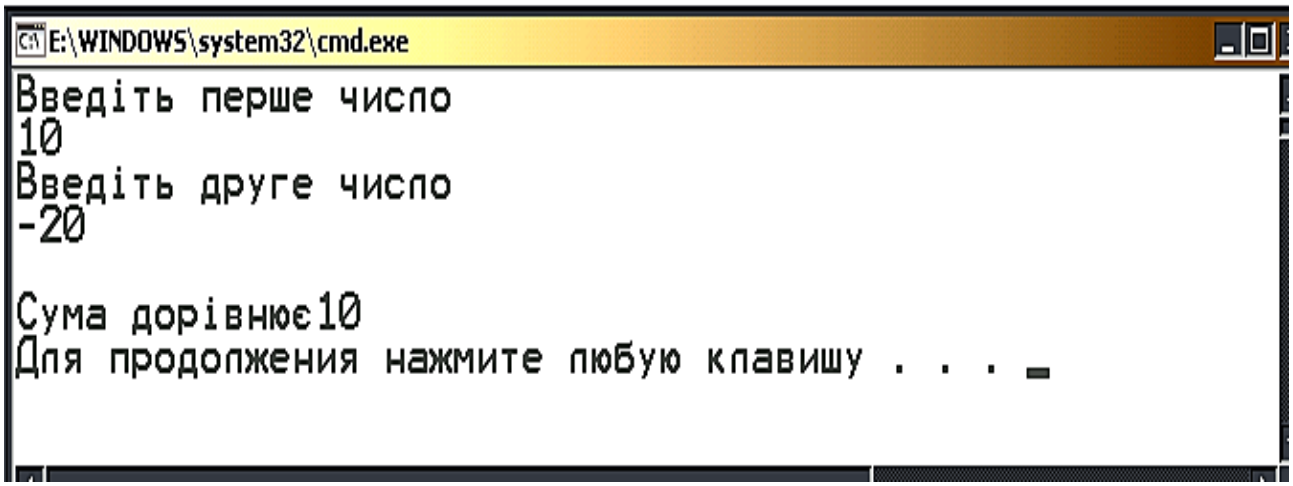
План тестування

Вхідні дані		Очікуваний результат	Отриманий результат	Відмітка про проходженні тесту
1-е число	2-е число			
10	-20	10	10	+
1 234	4 321	5 555		

Текст програми

```
#include <math.h>
#include <iostream>
int main()
{
    int a,b;
    std::cout << "Введіть перше число" << endl;
std::cin >> a;
std::cout << "Введіть друге число" << endl;
std::cin >> b;
    std::cout << "\nСума дорівнює" << abs(a+b);
    getchar();
    return 0;
}
```

Результат роботи програми



Висновки по роботі

У ході виконання лабораторної роботи № 1 було вивчено методику вирішення на комп'ютері завдань лінійного характеру, вивчено можливості мови С++ із введення інформації з клавіатури і виводі на екран, вивчено порядок роботи з математичними функціями (бібліопапки Math), отримано практичні навики роботи в середовищі Microsoft Visual Studio 2010.

Додаток Б

Середовище візуальної розробки програм Microsoft Visual Studio .NET

1. Середовище Integrated Development Environment

Середовище візуальної розробки програм Microsoft Visual Studio, зване також IDE (Integrated Development Environment – Інтегроване Середовище Розробки) призначене для створення, відкриття, перегляду, редагування, збереження, компіляції, побудови й відладки додатків, написаних на C або C++.

Запуск Microsoft Visual Studio .NET виконується за допомогою меню Пуск за стандартною схемою (рис. Б.1).

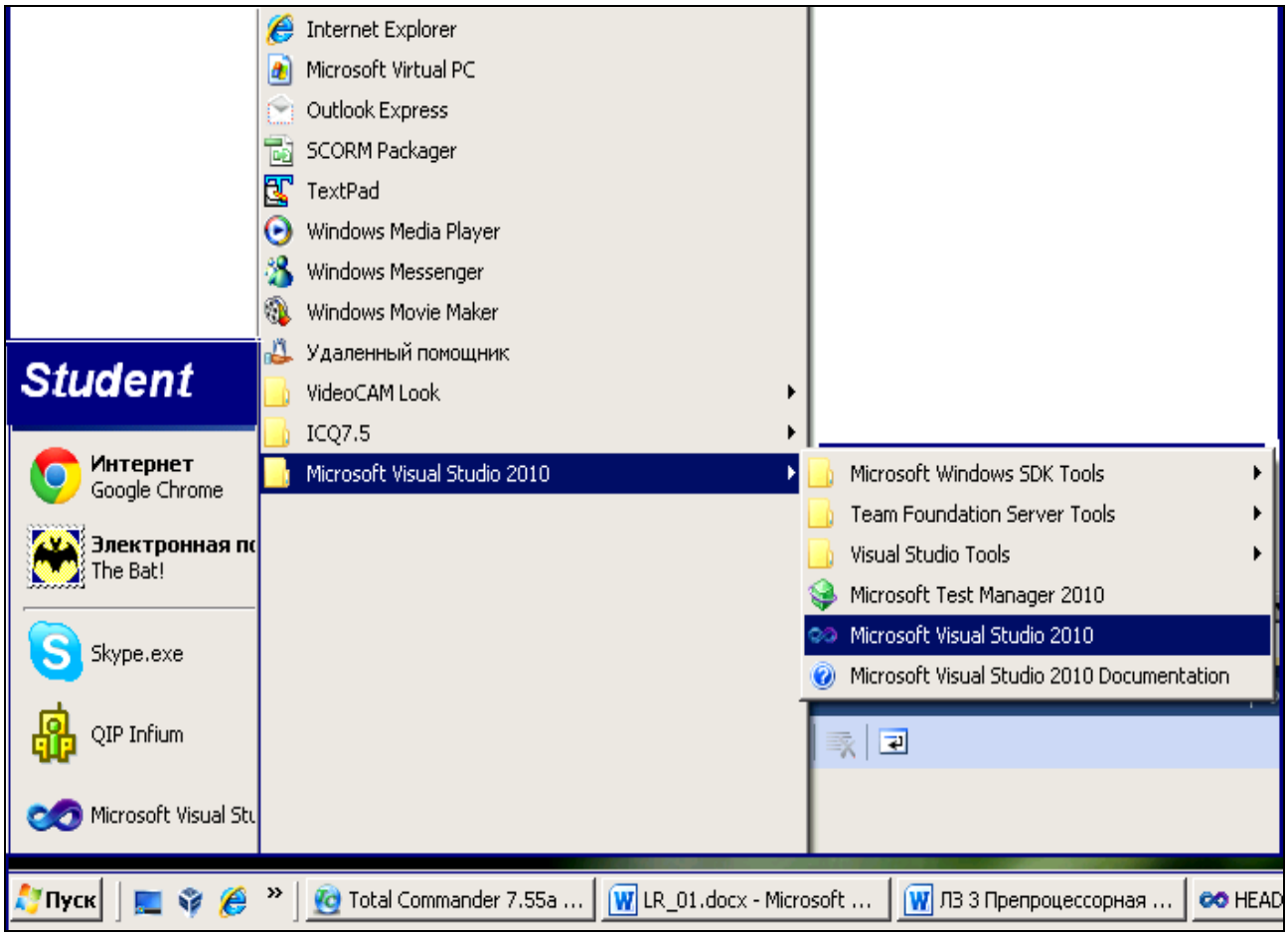


Рис. Б.1. **Запуск Microsoft Visual Studio .NET**

Після цього з'явитися головне вікно IDE із стартовою сторінкою (рис. Б.2).

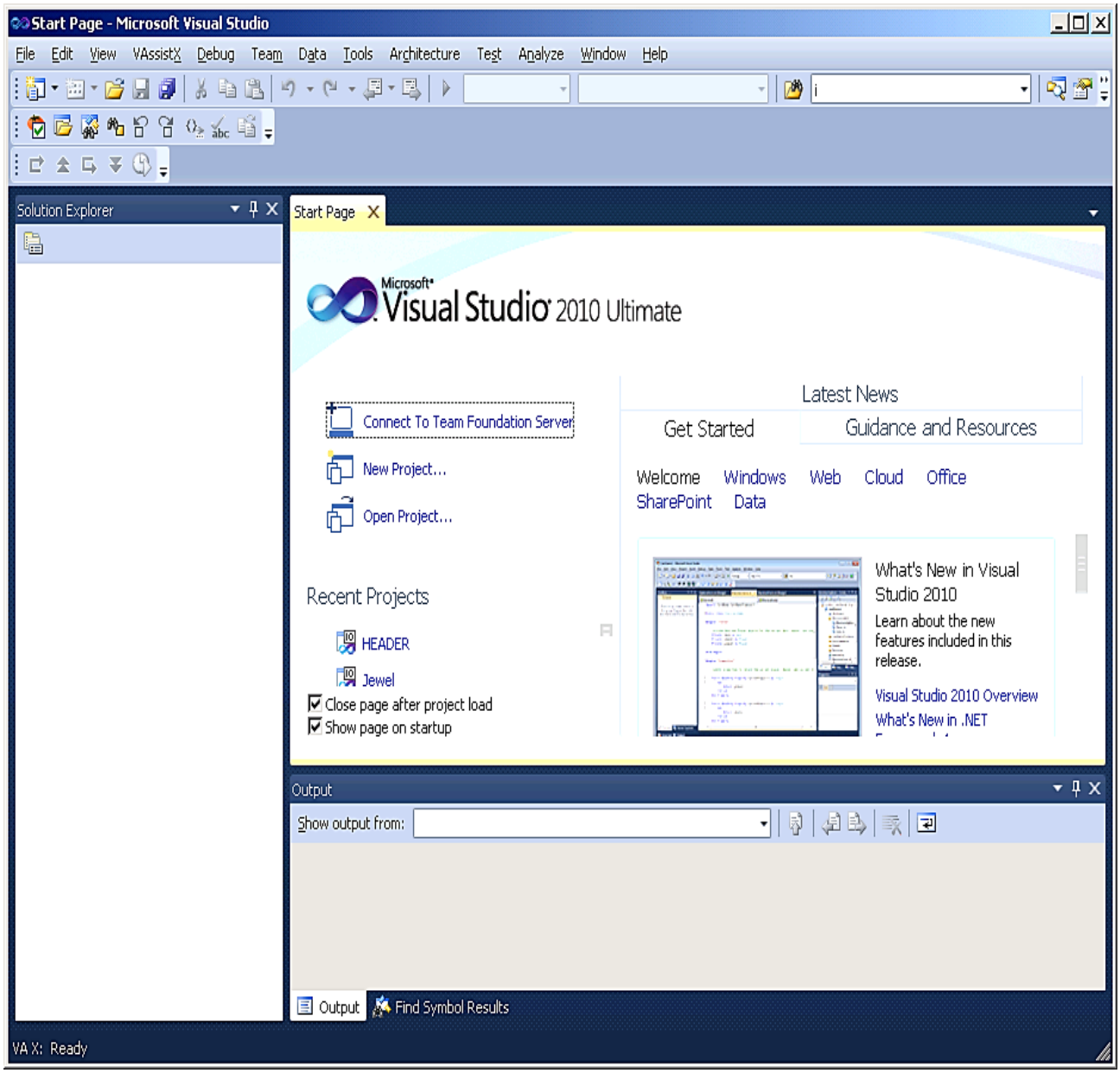


Рис. Б.2. **Стартова сторінка**

При певному налаштуванні середовища може з'явитися головне вікно IDE (рис. Б.3).

\

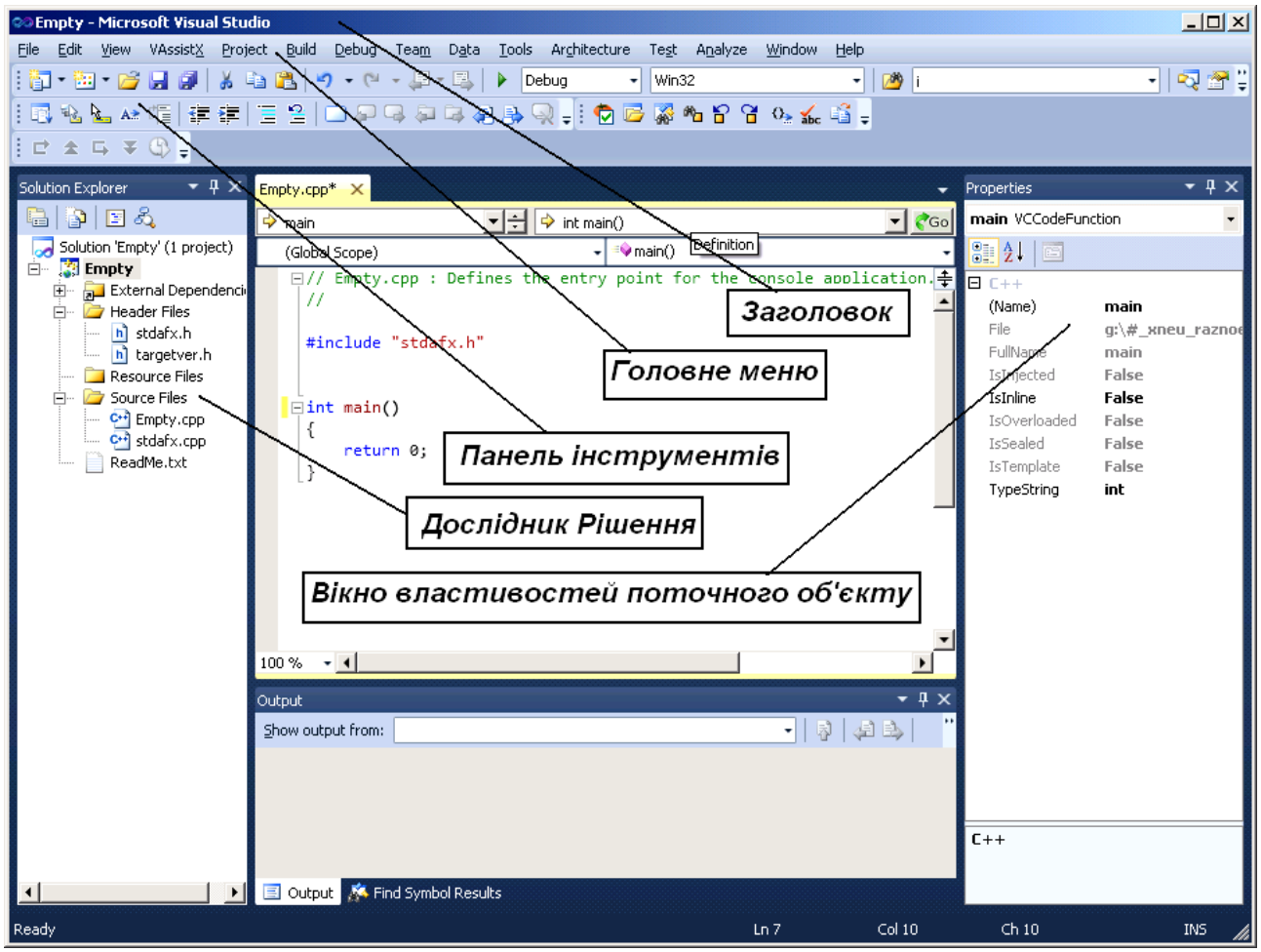


Рис. Б.3. **Головне вікно IDE**

2. Команди головного меню

Виклик команд меню виконується трьома способами:

за допомогою гарячих клавіш (вони указуються в меню праворуч від відповідного пункту);

за допомогою мишки (вказати й клацнути);

за допомогою відповідної кнопки (піктограми) на панелі інструментів (вони вказуються зліва від відповідної команди).

Активізація меню здійснюється за допомогою мишки (вказати й клацнути) або клавіш швидкого виклику (Alt + підкреслена буква в пункті меню). У випадних меню можна зустріти команди, виділені блідо-сірим кольором. Такі команди в нинішній момент часу недоступні через відсутність умов, необхідних для їх виконання.

Якщо за назвою команди меню виникає три крапки, це означає, що в результаті вибору даної команди буде відкрито діалогове вікно, якщо темний трикутник, то підменю.

Головне меню включає такі пункти:

1. File містить стандартний для багатьох Windows-додатків набір команд, призначених для маніпулювання файлами:

Команда	Призначення
New	Команда призначена для створення нового проекту
Open	Команда приводить до завантаження діалогового вікна Open File, за допомогою якого відкривається будь-який раніше збережений файл, або діалогового вікна Open Project, призначеного для відкриття проектів
Close	Команда призначена для закриття завантажених файлів
Add New Item Ctrl+Shift+A Add Existing Item Shift+Alt+A Add Project	Команди дозволяють створити файл нового проекту, відкрити існуючий проект, а також додати новий проект у поточний відповідно
Open Solution Close Solution	За допомогою цих команд відкривається існуюче рішення або закривається активне рішення
Save Ctrl+S	Збереження вмісту активного вікна
Save As	За допомогою цієї команди копія вмісту вікна може бути збережена у файлі під новим ім'ям
Save All Ctrl+Alt+S	Команда дозволяє зберегти всі відкриті на даний момент файли
Source Control	Команда приводить до відкриття підменю, яке призначене для порівняння властивостей файлу, для об'єднання окремих компонентів, встановлення зв'язку з базою даних, здійснення доступу до даних і контролю параметрів оновлення
Page Setup	Установка параметрів сторінки
Print Ctrl+P	Вивід на друк вмісту активного вікна
Recent Files Recent Project	Списки є засобом швидкого завантаження файлу або проекту
Exit	Вихід з IDE Visual C++

2. Edit містить команди, які дозволяють редагувати текст і проводити пошук за ключовими словами в програмному коді, що відображається в активному вікні. Ці команди ідентичні тим, що застосовуються в більшості текстових редакторів.

Команда	Призначення
Undo Ctrl+Z	Команда Undo дозволяє відмінити останню операцію редагування
Redo Ctrl+Y	Відновлює зміни, скасовані за допомогою команди Undo
Cut Ctrl+X	Команда дозволяє скопіювати виділений в активному вікні блок тексту в буфер обміну Clipboard і видалити його з вікна
Copy Ctrl+C	Команда копіює і поміщає виділений блок тексту в буфер обміну
Paste Ctrl+V	Команда здійснює вставку вмісту буфера в поточній позиції курсора
Cycle ClipBoard Ring Ctrl+Shift+V	Команда дозволяє вибрати необхідний фрагмент з буфера обміну і вставляє його в код
Delete Del	Команда застосовується для видалення символів або виділених фрагментів тексту в активному вікні
Select All Ctrl+A	Команда дозволяє виділити вміст активного вікна з метою подальшого вирізування, копіювання або видалення
Find and Replace	Команда за дією ідентична однойменному засобу пошуку, який надається в текстових редакторах
Go To…	Команда є засобом швидкого переходу до певного місця активного вікна
Insert File As Text	У результаті активізації команди відкривається вікно File Manager, що дозволяє вибрати текстовий файл, який можна скопіювати в активний документ
Bookmarks	Команда викликає підменю, яке дозволяє розмістити закладки в тих місцях програми, до яких ви звертаєтеся найчастіше
Outlining	Команда викликає підменю, яке дозволяє приховати вибраний фрагмент коду
Intellisence	Команда викликає підменю, яке дозволяє встановити режим контекстної довідки

3. View містить команди управління зовнішнім виглядом рішення, проекту, класів, коди, шаблонів і т. д.

За допомогою меню View можна отримати доступ до списку високого рівня, що містить назви всіх вікон і панелей, за допомогою яких на екран виводиться широкий спектр даних.

Команда	Призначення
Open Open With	Команда дозволяє відкрити файл у вибраному редакторові (наприклад, в двійковому редакторові Binary, редакторові ресурсів Resource і т. д.)
Solution Explorer Ctrl+Alt+L	Команда надає можливість виконувати різні операції з компонентами рішення: переносити файли з одного проекту в інший, виводити на екран властивості файлів і проектів, а також додавати в рішення як нові, так і існуючі файли або проекти
Class View Ctrl+Shift+C	Команда дозволяє виводити інформацію про вибраний клас
Server Explorer Ctrl+Alt+S	У вікні Server Explorer демонструються ресурси серверів, до яких можливий доступ по мережі з даного комп'ютера (наприклад, SQL-серверу)
Resource View Ctrl+Shift+E	Команда виводить на екран список усіх ресурсів активного додатка
Properties Window Alt+Enter	Команда надає детальну інформацію про властивості, які розподілені за категоріями і представлені в алфавітному порядку
ToolBox Ctrl+Alt+X	Вікно включає елементи управління періоду проектування, елементи управління ACTIVEX
Pending Checkins	Команда дозволяє сумісне використання оновленої коди початкових файлів декількома користувачами
Web Browser	Команда дозволяє вивести Web-сторінку на екран безпосередньо в IDE
Other Window	Команда є засобом проглядання макросів, об'єктів, схем документів, списків завдань і результатів пошуку. Вона також забезпечує проглядання вмісту вікна Command Window
Show Tasks	У результаті вибору однієї з команд цього підменю на екран виводиться список зареєстрованих завдань.
Toolbars	Дана команда служить для включення/відключення всіх доступних панелей інструментів.
Full Screen Shift+Alt+Enter	Активізація цієї команди приводить до збільшення сторінки з кодом до розмірів екрану, що спрощує його перегляд і редагування.
Navigate Backward Ctrl+- Navigate Forward Ctrl+Shift+-	Команди забезпечують переміщення вперед і назад по відкритих вікнах програми, а також повернення до змінених місць програми.
Property Pages	У вікні публікується детальна інформація про властивості, які розподілені за категоріями і представлені в алфавітному порядку

4. Project містить команди для управління відкритими проектами. Даний пункт головного меню з'явиться після створення проекту.

Команда	Призначення
Add Class Add Resource …… Add New Item Ctrl+Alt+A Add Existing Item Shift+Alt+A	Чотири перші команди меню використовуються для додавання в поточний проект компонентів, вказаних в їх назвах
New Folder	Команда дозволяє створювати нові підкаталоги для розміщення ресурсів проекту
Add Web Reference	Команда завантажує браузер, з якого інтерактивні Web-сервіси можна додати в додаток середовища Visual Studio .NET
Set as Starting Project	Команда використовується при розробці складних проектів

5. Build забезпечує доступ до елементів IDE, призначених для генерації коду, відладки і запуску створеної програми. Найбільш важливою в даному меню є команда Rebuild Solution. Даний пункт головного меню з'явиться після створення проекту.

Команда	Призначення
Build Solution F7	Команда досліджує всі файли проекту, після чого компілюються і компонуються тільки ті залежні файли, які були створені пізніше, ніж виконуваний файл проекту
Rebuild Solution	Відмінність між командами Build Solution і Rebuild Solution полягає в тому, що остання не враховує дату створення файлів і компілює й компонує всі файли проекту
Clean Solution	Команда дозволяє видалити всі файли з проміжних каталогів у будь-якій конфігурації робочого простору проекту
Build fp Rebuild fp Clean fp	Цей блок команд використовується для побудови вирішення файлу, а не проекту
Batch Build	Дана команда аналогічна команді Build Solution, але з її допомогою можна обробити відразу декілька конфігурацій одного проекту
Configuration Manager	Команда дозволяє змінити деякі параметри конфігурації скомпонованного блоку
Compile Ctrl+F7	Вибір даної команди служить вказівкою відкомпілювати вміст поточного вікна

6. Debug містить команди для налаштування відладчика. Даний пункт головного меню з'явиться після створення проекту.

Команда	Призначення
Windows	Команда дозволяє проглянути розставлені точки зупинки
Start F5	При виклику даної команди програма виконується вцілом або до точки зупинки
Start Without Debugging Ctrl+F5	Команда дозволяє задати безупинне виконання алгоритму без використання яких-небудь засобів відладки
Exception Ctrl+Alt+E	Команда викликає діалогове вікно Exceptions, яке містить ієрархічний список виключень, згрупованих за категоріями
Step Into F11 Step Over F10	Команди встановлюють покроковий режим роботи. Різниця між ними виявляється, коли відладчик викликає функцію або метод
New Breakpoint… Ctrl+B	Команда дозволяє задати позицію "точки останову" відладки, в якій відладчик зупиниться
Clear All Breakpoints Ctrl+Shift+F9	Команда видаляє всі точки зупинки за один крок

7. Tools містить команди, які дозволяють активізувати утиліти, необхідні для управління надзвичайно складними додатками, написаними мовою C++.

Команда	Призначення
Debug Processes… Ctrl+Alt+P	Дана команда подібна до команди Processes з меню Debug
Connect to Database…	Команда дозволяє визначити спосіб підключення до Microsoft SQL-серверу з середовища Visual Studio .NET для отримання даних
Add/Remove Toolbox Items…	Команда застосовується для додавання та видалення елементів управління, об'єктів і інших компонентів вікна Toolbox
Add-in Manager…	Команда дозволяє відобразити список вбудованих додаткових пристроїв
Build Comment Web Pages	Команда викликає діалогове вікно Build Comment Web Pages, на яких створюються сторінки htm із структурою коди, збереженої у файлах проектів, а також тексти коментарів

Команда	Призначення
Macros	За допомогою команд підменю Macros забезпечується повне управління макросами в IDE системи Visual C++. При записі макросу фіксуються вироблювані дії. На основі цієї інформації і створюється код макросу. Проте не всі дії, що виконуються за допомогою елементів призначеного для користувача інтерфейсу, можна записати в макросах
External Tools…	Команда використовується для додавання спеціальних команд в меню Tools. За допомогою цих команд з середовища розробки можна запускати зовнішні інструменти
Customize…	Дана стандартна команда дозволяє налаштовувати всі панелі інструментів IDE, додавати або видаляти кнопки виклику команд, а також проводити налаштування їх роботи
Options…	Команда використовується для зміни налаштувань IDE, заданих за замовчуванням

8. Window містить стандартний набір команд, характерний для всіх Windows-приложений. Команди меню View надають можливість визначити, в якому вигляді вікна мають бути виведені на екран.

Команда	Призначення
New Wondow	Після активізації команди New Window створюється копія файлу в новому вікні
Split	Команда Split поверх вікна Edit виводить чотири квадратні підвікна, в яких ви можете власноруч визначити положення горизонтальних і вертикальних ліній розділу
Dockable Hide Floating	Панель інструментів може бути закріплена з будь-якого краю батьківського вікна (команда Dockable), а може знаходитися у власному маленькому вікні і займати будь-яку позицію на екрані (команда Floating)
Close All Documents	Команда дозволяє закрити всі вікна, відкриті в середовищі IDE
1 fp.cpp	Щоб потрапити в одне з відкритих вікон, досить вибрати його назву в динамічному списку меню Window
Windows	Ця команда відкриває список активних на даний момент вікон

9. Help включає команди, які дозволяють звернутися до довідкових засобів середовища Visual Studio .NET і Dynamic Help.

Команда	Призначення
Dynamic Help Ctrl+F1	Команда забезпечує виклик динамічної довідки
Index results… Shift+Alt+F2 Search results… Shift+Alt+F3	Команди дозволяють виводити назви розділів, знайдених в наочному покажчику, і результати пошуку певних довідкових розділів
Previous topic Next topic Sync Contents	Перераховані в заголовку команди доступні, якщо у вікні Edit міститься довідкова інформація
Show Start Page	Команда виводить на екран зміст початкової стартової сторінки середовища Visual Studio .NET, на якій можна дістати доступ до новинок, пов'язаних з Visual Studio .NET
Check for Updates	Команда виводить інформацію про останні розробки компанії Microsoft для середовища Visual Studio .NET
Technical Support	Команда забезпечує зв'язок з компанією Microsoft з питань, що стосуються середовища Visual Studio
Help on Help	Команда дозволяє отримати інформацію про способи використання складних команд контекстно-залежної довідки середовища Visual Studio .NET
About Microsoft Visual Studio	У стандартному вікні About відображається інформація про версію програми, її ідентифікаційний номер і встановлені компоненти

3. Створення проекту

Створення нового проекту здійснюється за допомогою пунктів системи меню (рис. А.4) середовища IDE або за допомогою гарячих клавіш Ctrl+Shift+N.

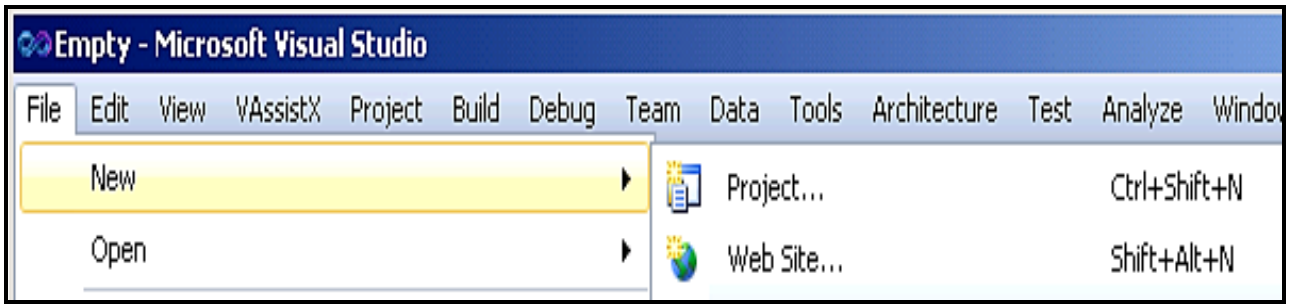


Рис. Б.4. **Виклик діалогового вікна New Project.**

Після виконання вказаних дій з'явитися діалогове вікно New Project (рис. Б.5).

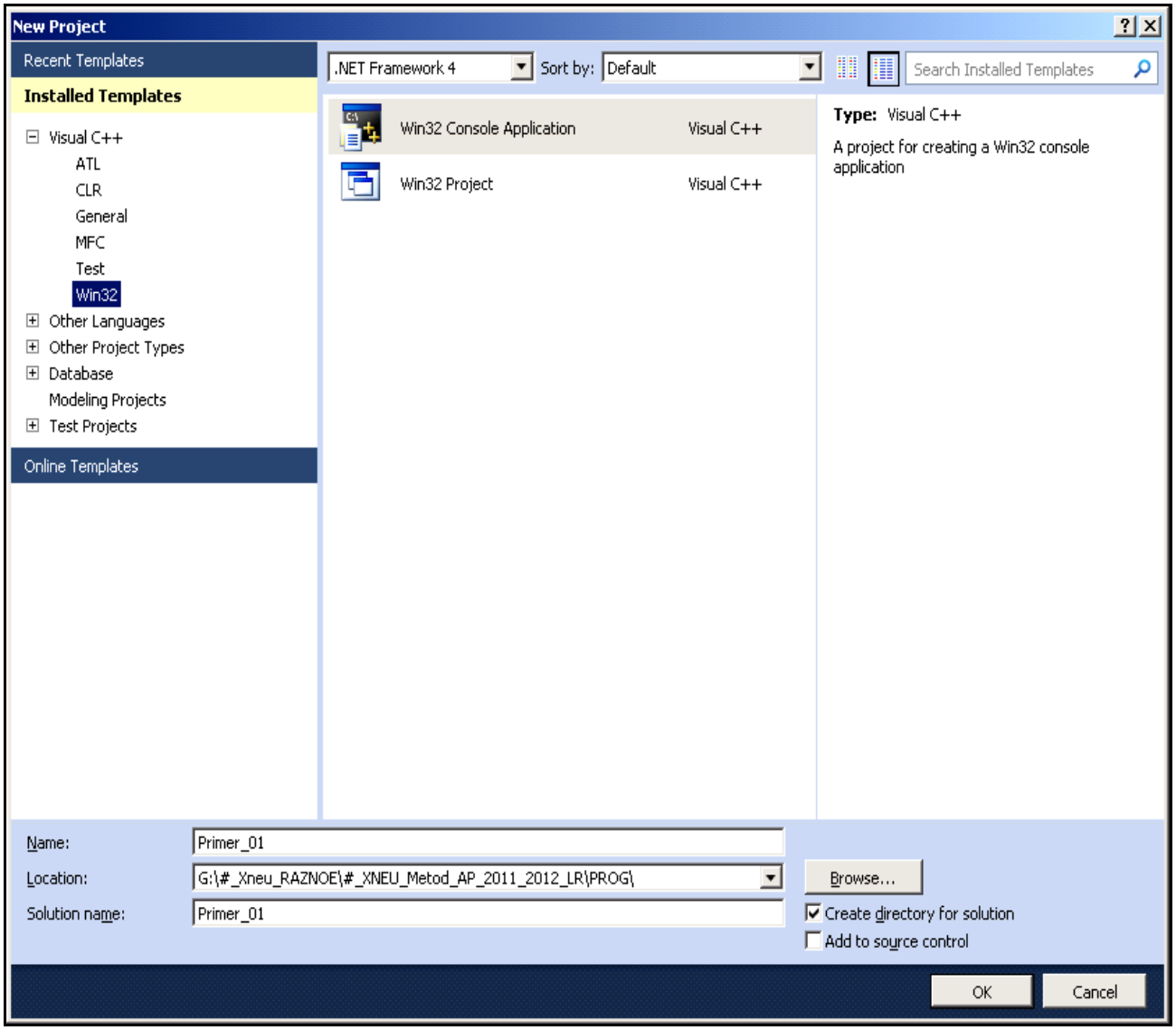


Рис. Б.5. **Діалогове вікно New Project**

У цьому вікні потрібно вибрати:

мову і тип проекту;

ім'я проекту (наприклад Primer_01);

директорію, в яку буде поміщений новий проект;

.

Після клацання на кнопці ОК на екрані відобразиться вікно майстра (рис. Б.6).

Рис. Б.6. **Вікно майстра створення проекту.**

Якщо натиснути кнопку Finish, то проект створиться за замовчуванням, а якщо Next, то перейдемо до вікна Application Setting, на якому треба вибрати установки створюваного проекту (рис. Б.7).

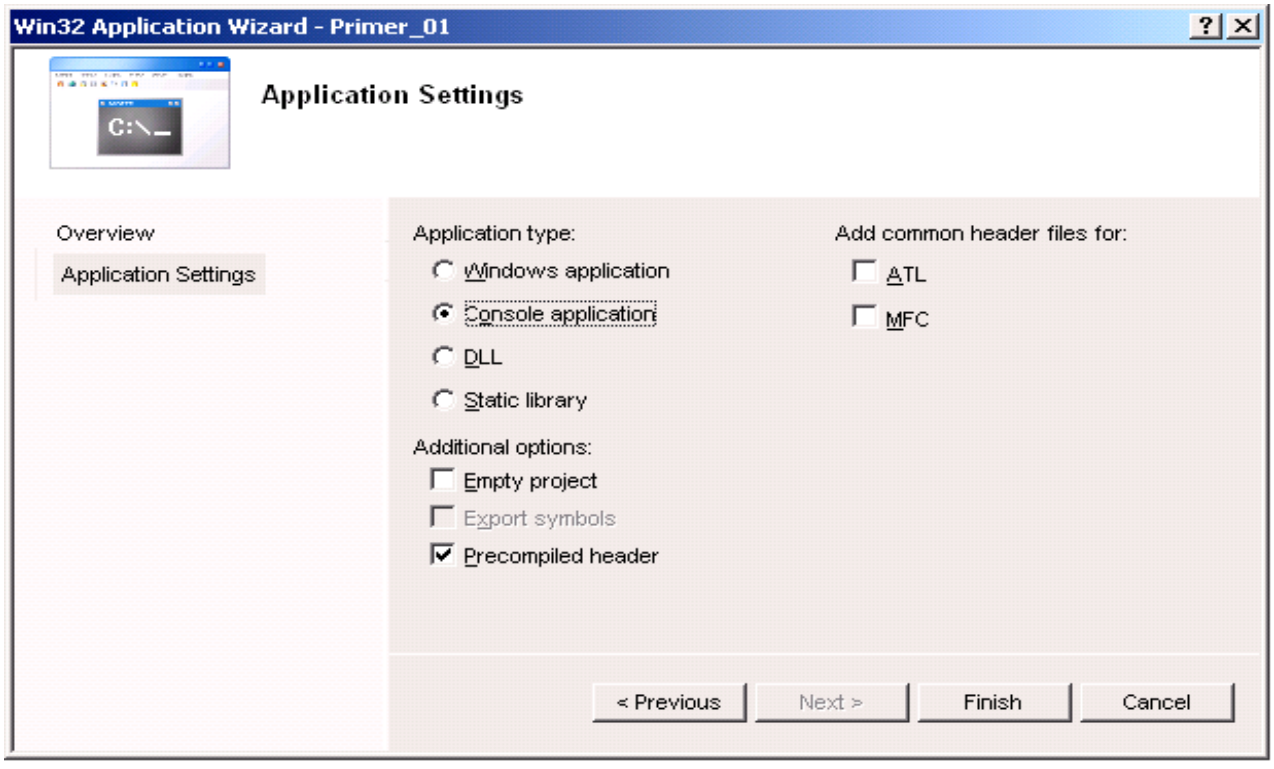


Рис. Б.7. **Вікно майстра створення додатка Win32**

Для створення простого Windows-додатка необхідно включити перемикач Console Application і встановити прапорець Empry project.

Після натиснення кнопки Finish в зовнішній пам'яті у вибраній директорії буде створена піддиректорія (каталог, папка) проекту Primer_01 і на екрані дисплея з'явиться головне вікно IDE з темною робочою областю. У рядку головного меню з'явиться три додаткові пункти (Project, Build, Debug).

Далі необхідно створити файл початкового тексту програми. Для цього необхідно виділити теку Source Files у вікні провідника рішення (Solution Explorer). У контекстному меню вікна треба вибрати команду Add і далі в додатковому меню, що з'явилося, вибрати команду Add New Item (рис. Б.8).

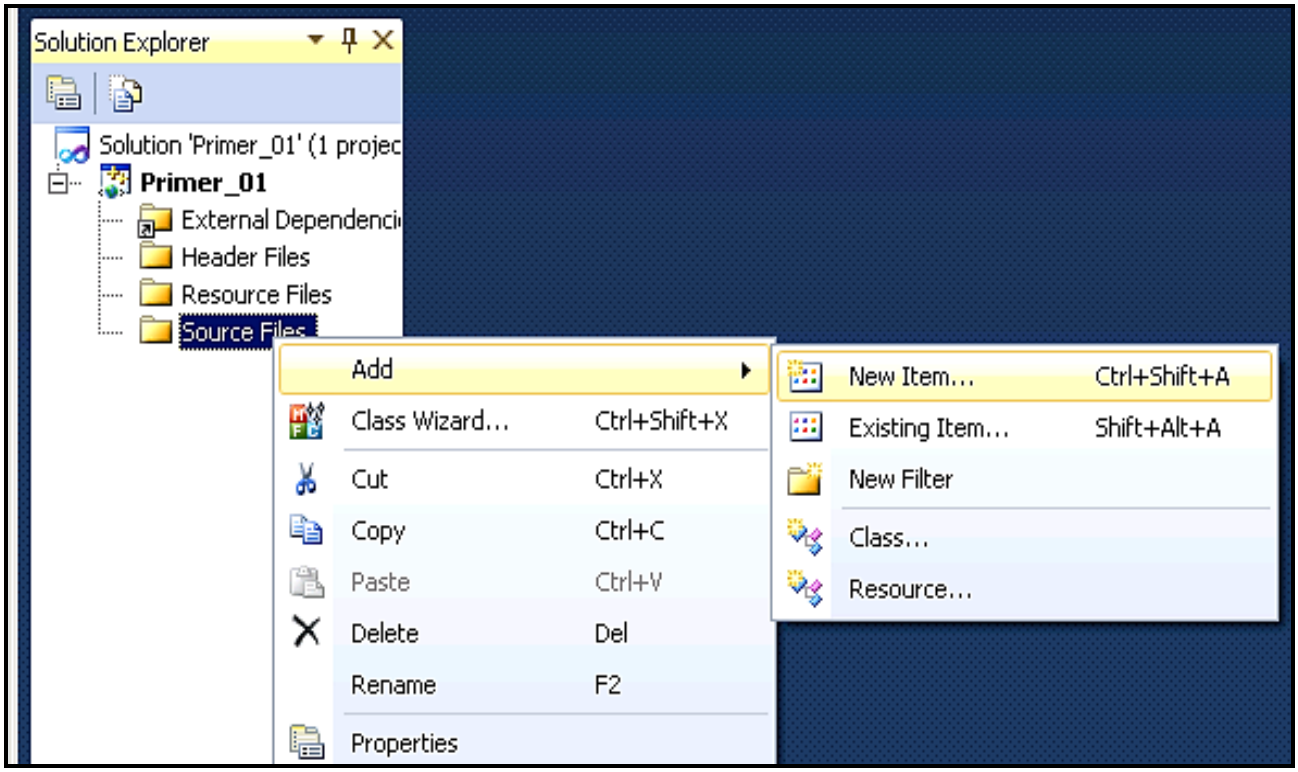


Рис. Б.8. **Процес створення файлу початкового тексту програми**

У результаті виконаних дій з'явитися діалогове вікно Add New Item (рис. Б.9). У вікні треба вказати ім'я файлу і вибрати шаблон (Templates).

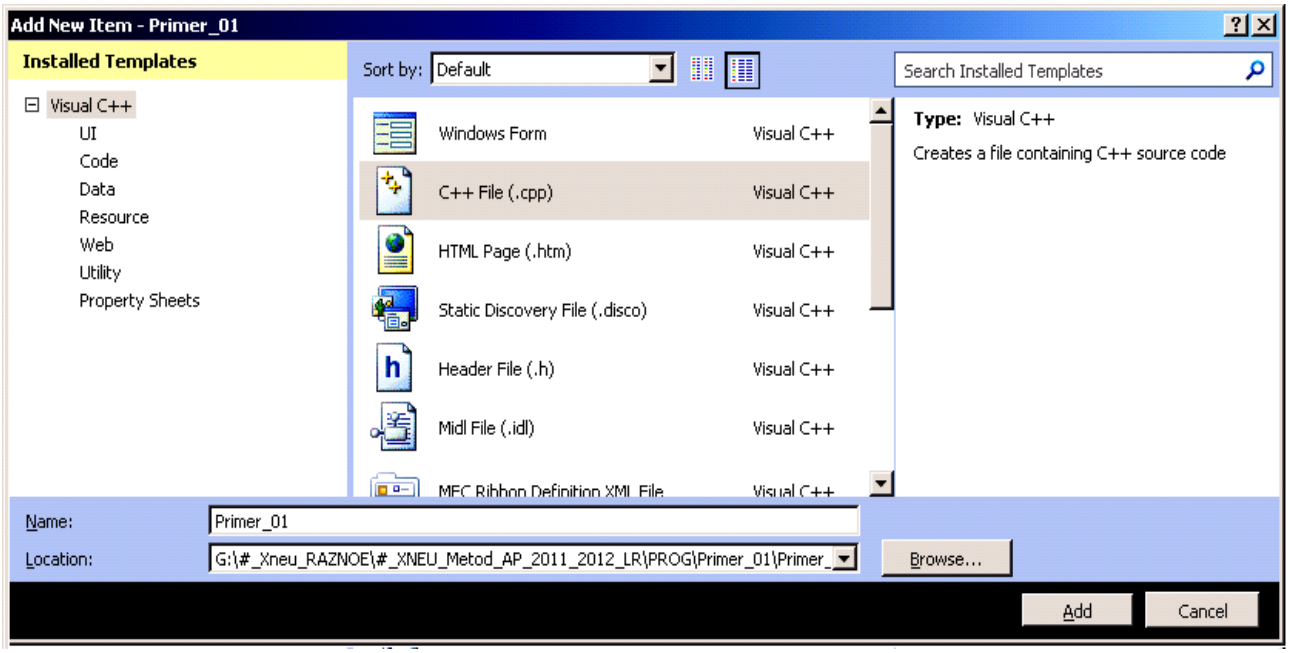


Рис. Б.9. **Діалогове вікно Add New Item**

Після натиснення кнопки Open в папці проекту буде створений файл початкового тексту програми Primer_01.cpp і активізується редактор початкового тексту програми. У заголовку головного вікна IDE з'явиться ім'я файлу. Після цього можна ввести початкові команди створюваного проекту (рис. Б.10).

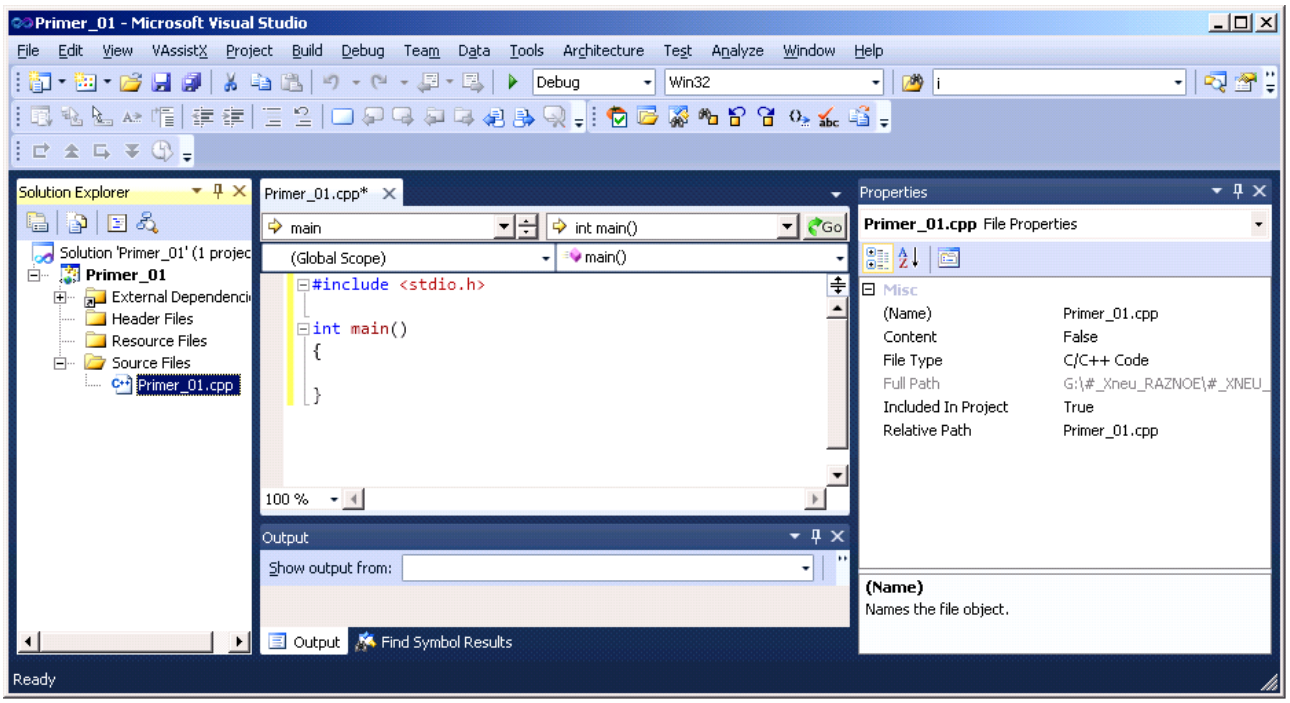


Рис. Б.10. **Введення початкового тексту програми**

4. Введення й редагування початкового тексту програми

Введення й редагування початкового тексту програми виконується за правилами, які використовуються в текстових редакторах.

Внесення виправлень до тексту

Невеликі зміни вносяться до тексту за допомогою клавіш [Backspace] або [Del] або введенням виправленого тексту в режимі заміни. У разі "великомасштабних" змін використовуються команди меню Edit: Cut, Copy і Paste (для цих операцій передбачені поєднання клавіш і кнопки в інструментальній панелі Standard і контекстному меню).

Друкарські помилки в процесі введення тексту виправляються за допомогою клавіші [Backspace]. При цьому віддаляються символи, що стоять зліва від курсора. [Ctrl]+[Backsapce] дозволяють видалити ціле слово зліва від курсора.

Натиснення клавіші [Ins] або подвійне клацання на полі INS в рядку стану активізує режим заміни. У рядку стану з'являється індикація поля OVR. Зазвичай редактор за замовчуванням працює в режимі вставки. Це означає, що символ вставляється в існуючий текст, а текст, розташований праворуч від курсора, зрушується вправо.

Для вставки тексту слід перевести курсор у потрібну позицію, використовуючи навігаційні клавіші або мишу, і набрати текст, що вставляється.

Для відміни введення або видалення необхідно натиснути [Alt]+[Backspace] або вибрати команду меню Edit>Undo.

Переміщення курсора, виділення тексту

Курсор введення (мерехтливий вертикальний штрих) позначає поточну позицію введення тексту.

За допомогою клацання миші в деякій позиції документа можна помістити курсор введення в цю позицію. При роботі з клавіатурою окрім навігаційних клавіш (клавіш управління курсором) для переміщення курсора в розпорядження користувача надані спеціальні поєднання клавіш (табл. Б.1).

Таблиця Б.1

Клавіші для переміщення курсора

Клавіша	Переводить курсор...
[стрілка вліво], [стрілка вправо]	...на один символ праворуч або ліворуч
[стрілка вгору], [стрілка вниз]	...на один рядок вгору або вниз
[Ctrl]+[стрілка вправо]	...у позицію перед першим символом наступного слова
[Ctrl]+[стрілка вліво]	...у позицію перед першим символом попереднього слова
[Home]	...у початок поточного рядка
[End]	...у кінець поточного рядка
[Ctrl]+[Home]	...у початок програми
[Ctrl]+[End]	...у кінець програми

Прокручування зображення

Прокручування тексту у вікні за допомогою смуг прокрутки дозволяє переміщати вікно над документом, залишаючи незмінним положення курсора. Позицію курсора можна визначити за виведеною в рядку стану інформацією (Ln ... Col ...).

Якщо клацнути на кнопці-стрілці смуги прокрутки, то вміст вікна документа зміщується по рядкам вгору або вниз.

Якщо клацнути над бігунком (під бігунком), вікно переміщується над документом на один екран вгору (вниз).

Переміщати екран над текстом програми за бігунок.

Горизонтальна прокрутка виконується аналогічно вертикальній, з тією відмінністю, що вміст екрану зміщується праворуч або ліворуч.

Горизонтальна прокрутка можлива (і необхідна) лише в тому випадку, якщо довжина текстових рядків перевищує ширину екрану.

Виділення тексту

Для виконання більшості команд редагування або внесення змін у текст програми необхідно заздалегідь відповідним чином виділити фрагмент, що підлягає обробці.

Виділення найпростіше виконується за допомогою миші, проте і за допомогою клавіатури можна виділити фрагмент тексту посимвольно, послівно, позиція за позицією.

Виділення тексту (розширення виділення) можна виконати, натискаючи навігаційну клавішу спільно з клавішею [Shift] або [Shift]+[Ctrl]. Так, наприклад, [Shift]+[Ctrl]+[стрілка вправо]/[стрілка вліво] забезпечують розширення виділення тексту послівно [Shift]+[Home] – до початку рядка і т. д.

Виділення тексту за допомогою миші

Щоб виділити текст програми за допомогою миші, треба помістити курсор миші на перший символ виділення, натиснути ліву кнопку миші і, утримуючи її натиснутою, протягнути мишу в потрібному напрямі. По досягненню кінця фрагмента, що виділяється, відпустити кнопку миші.

Для виділення слова потрібно двічі клацнути на ньому лівою кнопкою миші.

Для виділення рядка потрібно клацнути лівою кнопкою миші навпроти рядка, що виділяється.

Копіювання, перенесення й видалення

Ефективність процесу редагування тексту програми можна підвищити, якщо скористатися вбудованим засобами копіювання, видалення й вставки.

Буфер обміну є постійно доступною під час робочого сеансу областю пам'яті, до якої можуть звертатися всі програми, що працюють в середовищі Windows. У буфер обміну переносяться, копіюються, а потім витягуються з нього для вставки фрагменти програми.

Остання вирізана або скопійована порція тексту зберігається в буфері обміну.

Вміст буфера обміну можна вставити в текст програми в декількох позиціях довільне число разів.

Швидке переміщення і копіювання фрагмента програми

Фрагменти тексту можуть бути перенесені і без використання буфера обміну. Для цього потрібно виділити переміщуваний текст. Помістити покажчик миші у виділений текст, натиснути і утримувати натиснутою ліву кнопку миші. Помістите покажчик миші в ту позицію, куди слід перемістити виділений текст, і лише після цього відпустити кнопку миші.

Для копіювання фрагмента тексту програми потрібно його виділити.

Помістити покажчик миші у виділений текст, натиснути і утримувати натиснутими клавішу [Ctrl] і ліву кнопку миші. Покажчик миші змінює свій

зовнішній вигляд, і він доповнюється прямокутником і знаком "+". Перемістити покажчик миші в ту позицію, куди слід скопіювати виділений фрагмент, і лише після цього відпустити кнопку миші.

Пошук і заміна

У редакторові початкового тексту програми передбачена можливість пошуку символів, слів або позицій, в яких був застосований вказаний формат. Знайдені елементи можна автоматично замінити іншими.

Для пошуку і заміни потрібно виконати команду Edit>Find and Replace або натиснути [Ctrl]+[F].Открывается діалогове вікно Find and Replace (рис. Б.11).

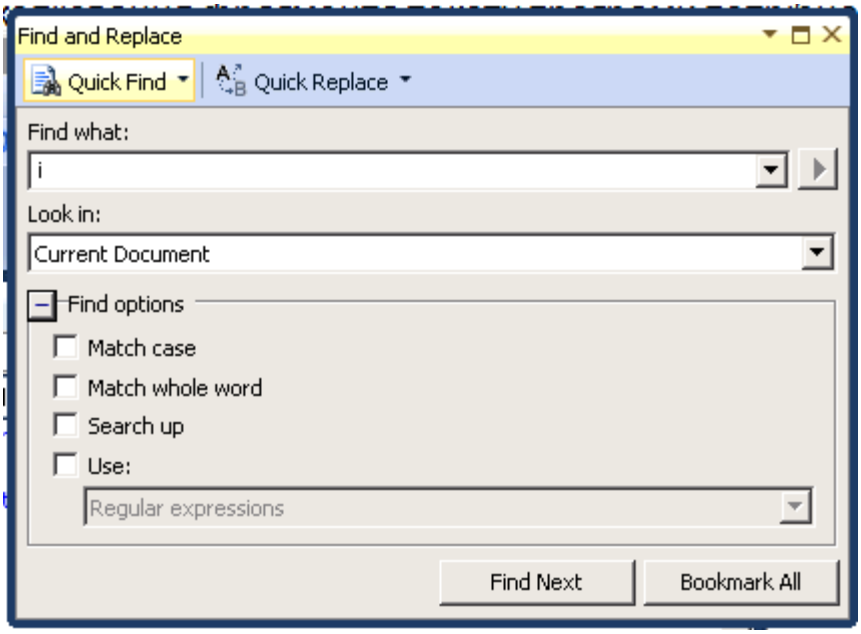


Рис. Б.11. **Діалогове вікно Find and Replace**

У полі Find what потрібно ввести текстовій фрагмент, який має бути знайдений (зразок). Його довжина не повинна перевищувати 255 символів.

Пошук можна виконувати

- у поточному документі;
- у всіх відкритих документах;
- у поточному проекті;
- у виділеному блоці поточного документа.

Вказані можливості пошуку і заміни важливі при роботі з багато-файловими проектами.

Для збереження тільки що введеного початкового тексту програми, потрібно скористатися командою меню File>Save або натиснути поєднання клавіш Ctrl+S.

На рис. Б.12 показано вікно редагування до збереження файлу. За замовчуванням файли автоматично зберігаються при виконанні команди Build Solution або Rebuild Solution. При виході з редактора, він запропонує зберегти результати роботи.

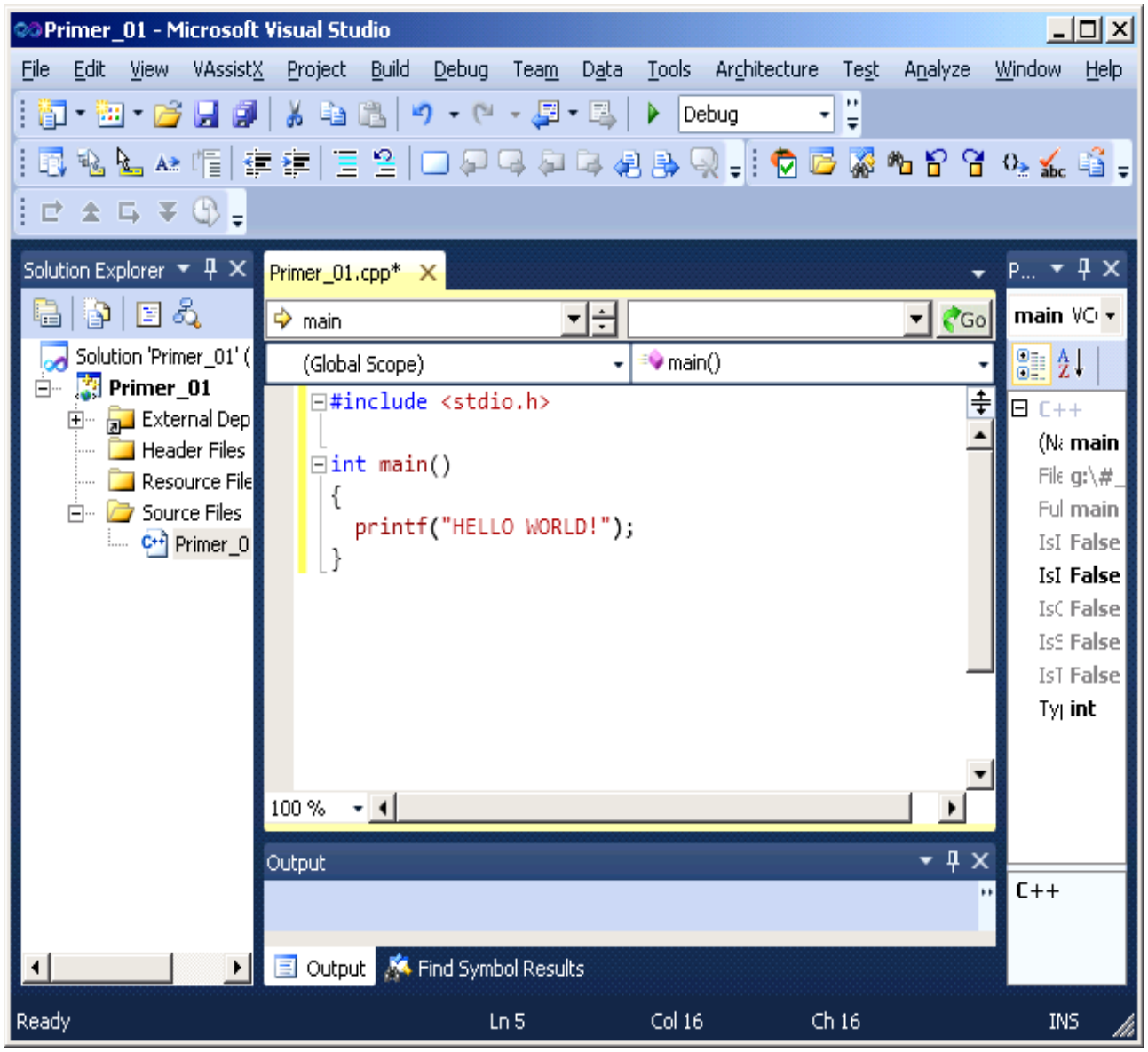


Рис. Б.12. **Вікно IDE з текстом програми**

5. Створення виконуваного файлу

Для створення виконуваного файлу рекомендується вико рис-товувати команду меню Build> Rebuild Solution.

Якщо в програмі були допущені синтаксичні помилки, то пові-домлення про них відображатимуться на вкладці Build вікна Output (рис. Б.13).

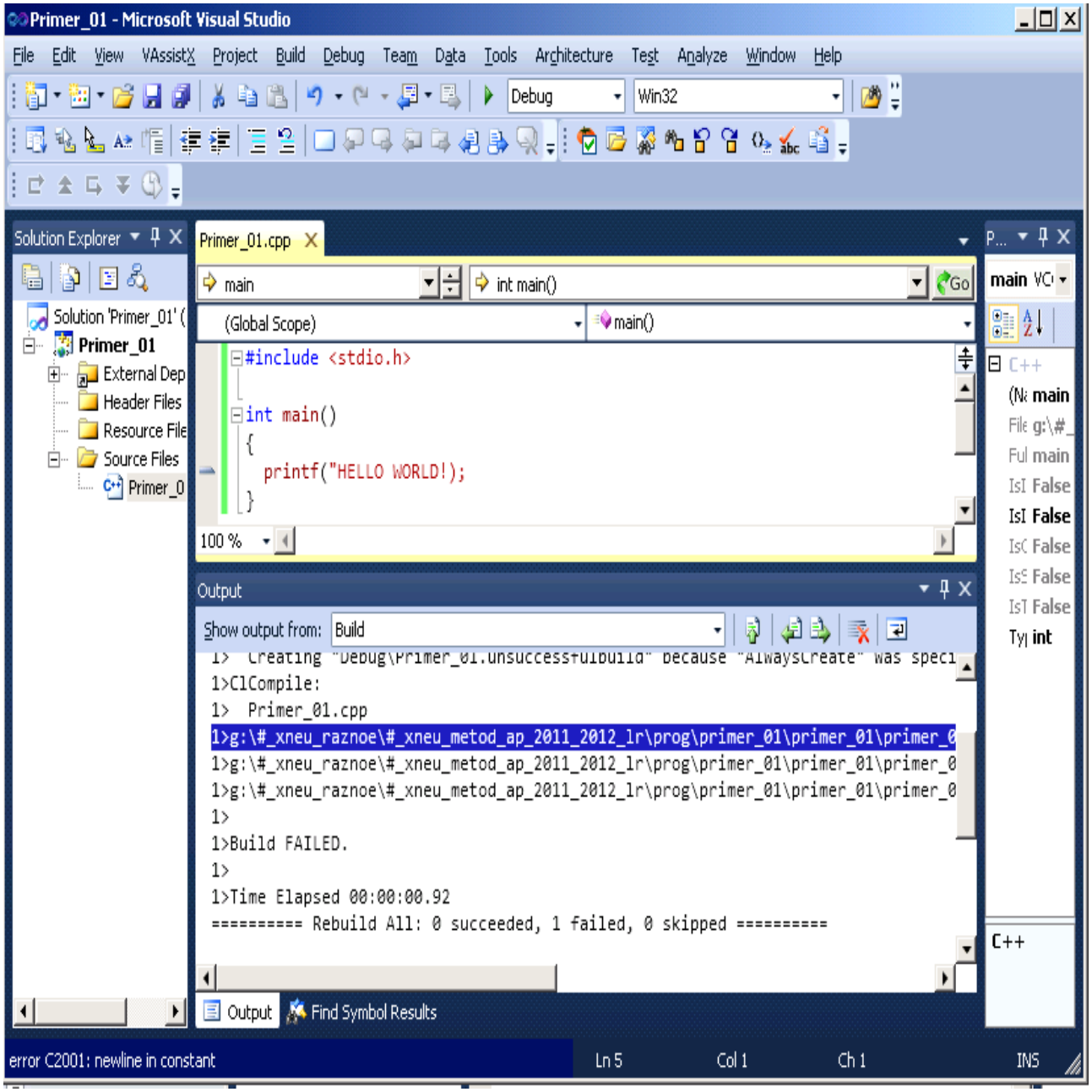


Рис. Б.13. **Вікно Output з повідомленням про помилки**

Повідомлення про помилку включає:

- номер помилки;
- категорію помилки (warning – попередження, error – помилка);
- короткий опис;
- ім'я файлу;
- номер терміну, в якому міститься помилка або попередження.

У даному прикладі суть помилки, що знаходиться в п'ятому рядку, полягає в тому, що забули поставити лапки у текстовій константі.

Застережливі повідомлення можуть з'являтися в тих випадках, коли компілятор автоматично виконує деякі стандартні перетворення і повідомляє про це програміста. Наприклад, якщо змінною типу int (ціле число) привласнюється дробове значення, то автоматично відбувається округлення. Це не означає, що в програмі допущена помилка, але оскільки перетворення типів даних виконується непомітно для програміста, компілятор вважає своїм обов'язком повідомити про це. Більшість функцій, оголошених у файлі math.h, приймають аргументи і повертають значення типу double (дійсне число подвійної точності). Якщо програма передасть одній з таких функцій аргумент типу float (дійсне число одинарної точності), компілятор, перш ніж направити дані в стек аргументів функції, виведе застережливе повідомлення про те, що тип даних float перетворений в double.

Для виявлення рядка в початковому тексті програми, яка викликала помилку, необхідно встановити курсор на рядок повідомлення і натиснути клавішу Enter або виконати подвійне клацання мишкою. Курсор у вікні редагування буде автоматично поміщений в рядок програми, що викликав появу повідомлення про помилку, а зліва від рядка з'явиться покажчик (рис. Б.13).

Після усунення помилки можна повторно використовувати команду меню Build> Rebuild Solution (рис. Б.14).

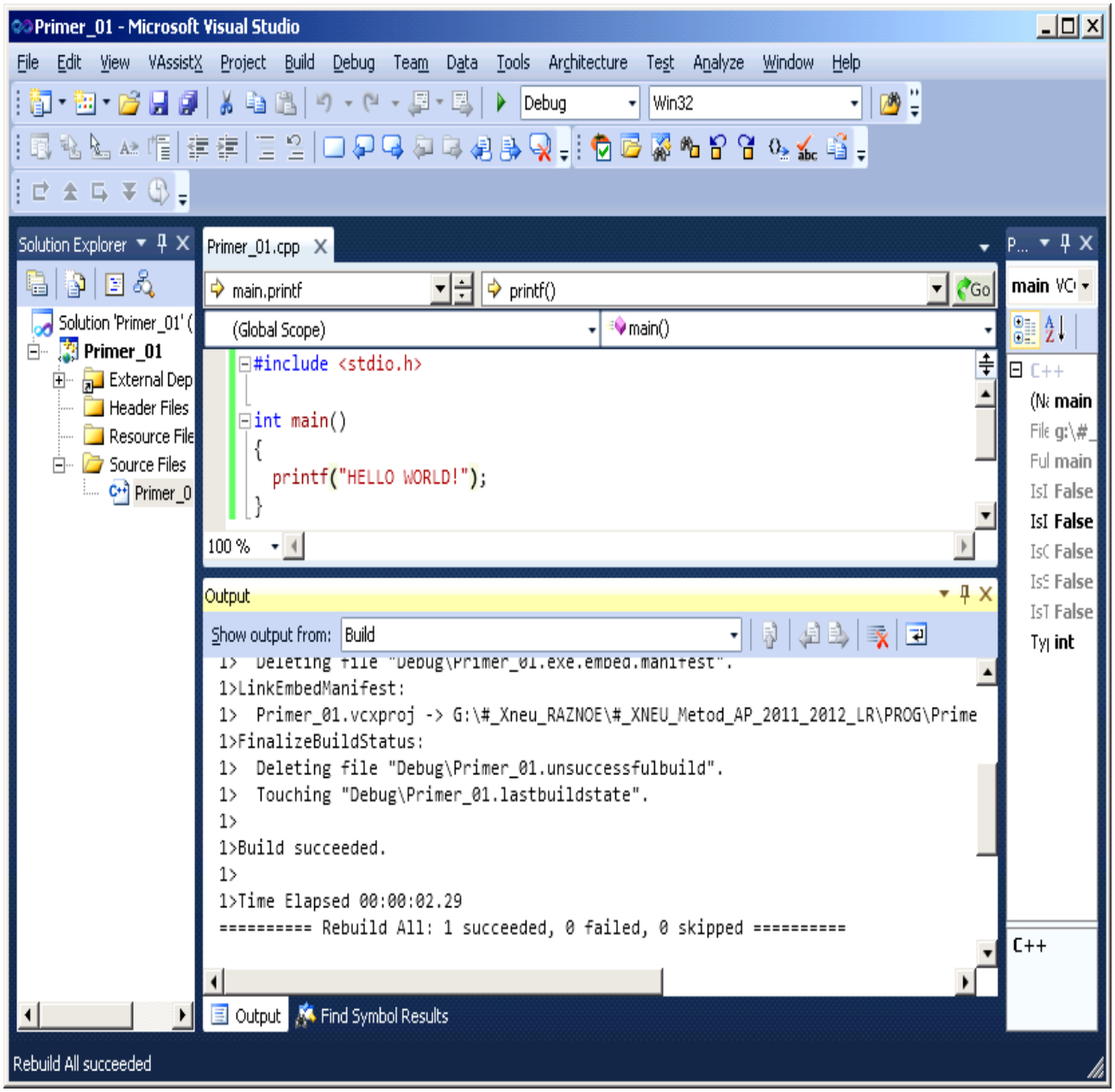


Рис. Б.14. **Вікно Output повторного сеансу побудови проекту**

Якщо проаналізувати вміст вікна Output, то можна побачити, що процес побудови виконавчого модуля програми включає трансляцію (Compiling) початкового тексту програми і скріплення (Linking) всіх складових проекту у виконуваний модуль. Результатом роботи транслятора і компонувальника є файли, що мають розширення *.obj і *.exe відповідно. Помилки можуть виникати як на етапі трансляції, так і на етапі побудови виконуваного модуля.

6. Запуск і відладка програми

Після успішної побудови виконуваного модуля, його можна запустити на виконання. Для цього, слід активізувати в меню Debug команду Start Without Debugging або натиснути Ctrl+F5. Якщо на етапі виконання не виникло помилок, то на екрані дисплея з'явиться консольне вікно з результатом (рис. Б.15).

Рис. Б.15. **Результат роботи програми**

Помилки, що виникають на етапі виконання, усуваються засобами відладки середовища IDE. Вони включають точки зупинки, вікна попереднього перегляду стану змінних і покрокове виконання програми.

Для демонстрації засобів відладки, внесемо невелику зміну до початкового тексту програми

```
#include <stdio.h>
int main()
{
  int A=2;
  int B=5;
  int C,D;
  printf("TEST BREAKPOINTS!\n");
  C=A+B;
  D=A*B;
  printf("C=%d  D=%d \n",C,D);
}
```

і створимо точку зупинки на дев'ятому рядку програми (рис. Б.16).

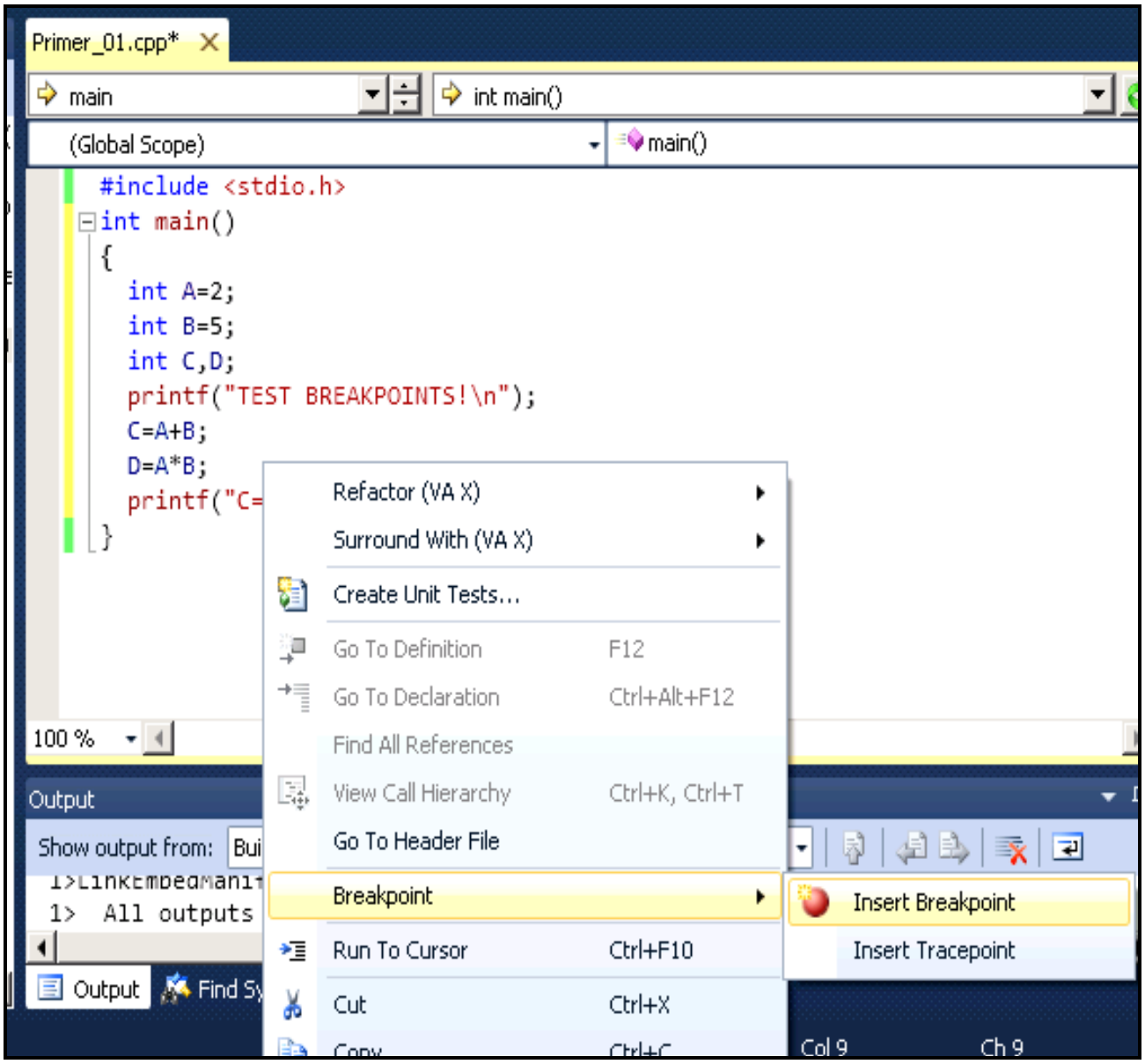


Рис. Б.16. **Створення точки зупинки**

Точки зупинки створюються за допомогою діалогового вікна New Breakpoints, що викликається командою New Breakpoints меню Debug або контекстного меню. Наявність кольорового кружка на початку рядка говорить про наявність точки останову в цьому місці (рис. Б.17).

Після розміщення точок зупинки можна активізувати команду Start меню Debug або натиснути F5. На рядку, де розміщена точка зупинки, виконання програми урветься. Далі за допомогою команди Windows меню Debug можна викликати вікна Autos або Locals і проглянути вміст змінних програми (рис. Б.18).

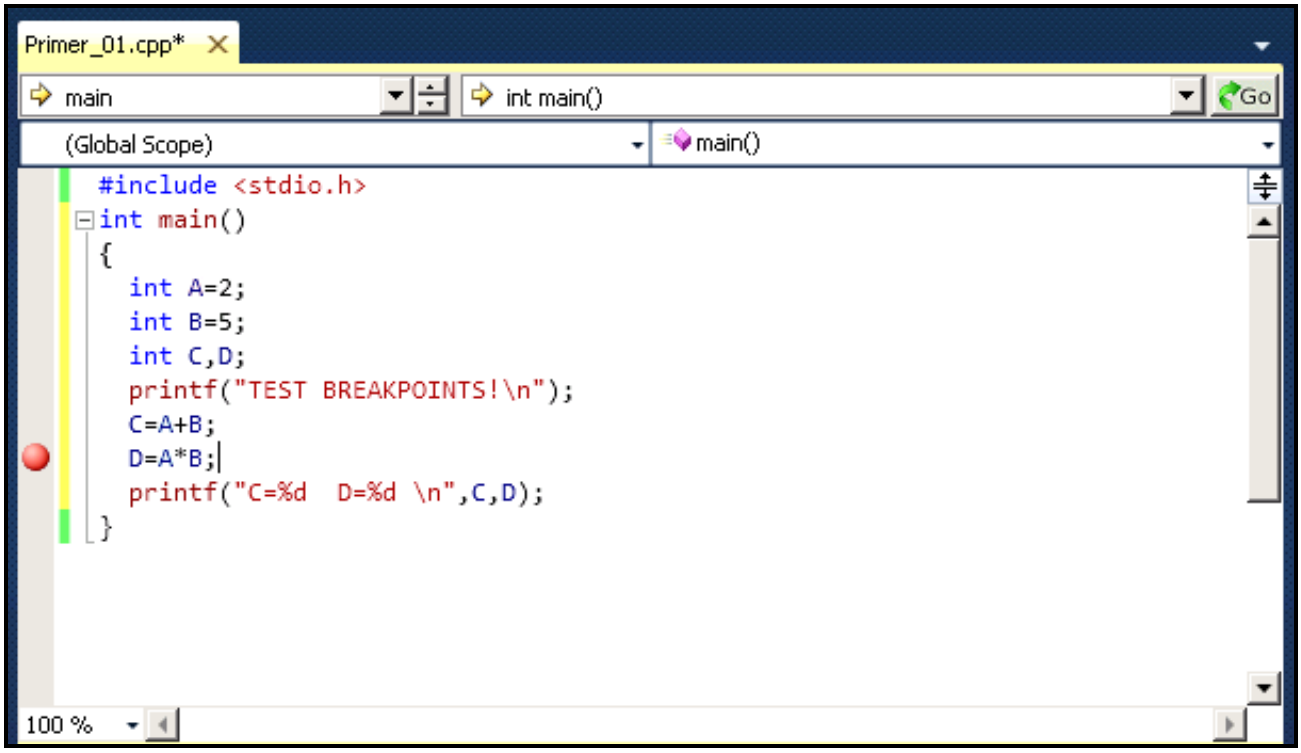


Рис. Б.17. **Текст програми з точкою зупинки**

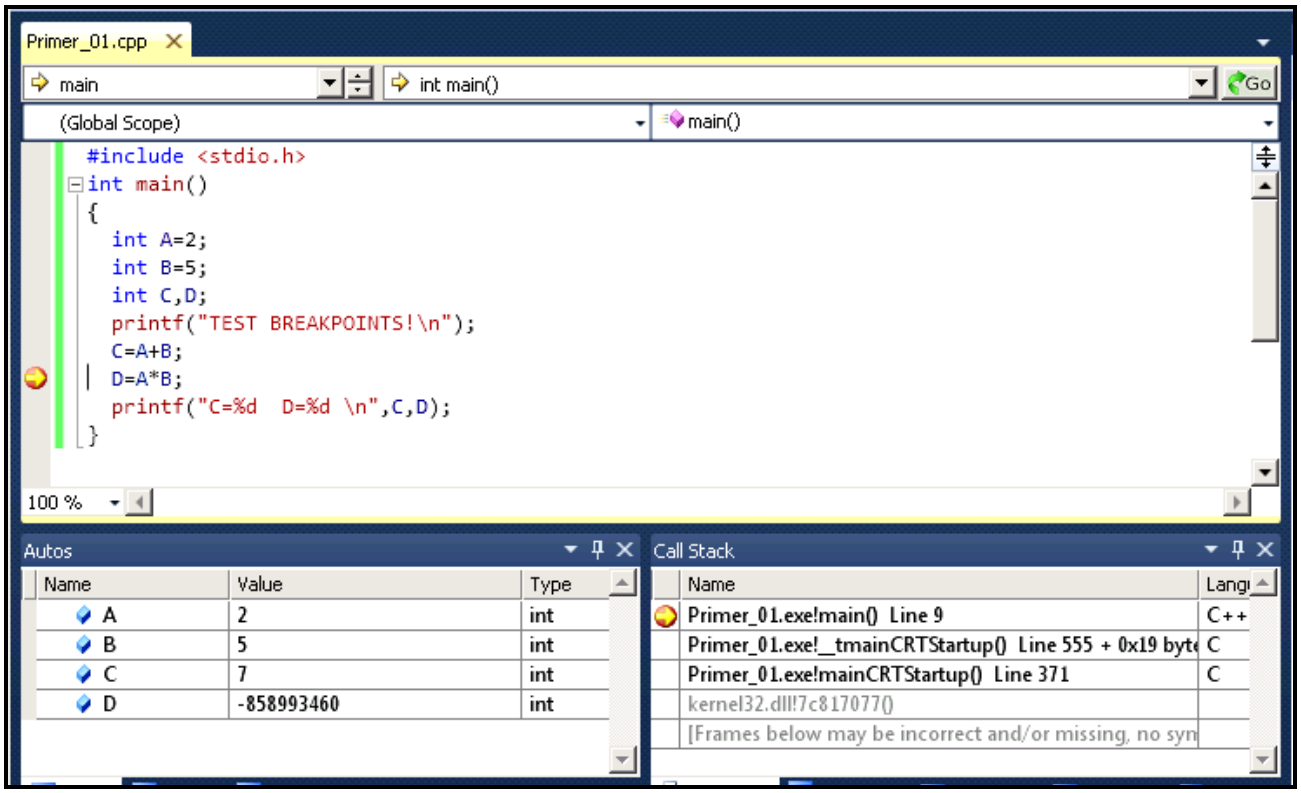


Рис. Б.18. **Перегляд значень змінних програми**

Далі виконання програми можна продовжувати в покроковому режимі. Покроковий режим виконання програми супроводиться покажчиком трасування (контурна стрілка на початку рядка) і активізується командами Step Into (F11) і Step Over (F10) меню Debug. Відмінності між командами з'являються тільки тоді, коли в програмі зустрічається виклик функцій. Якщо вибрати команду Step Into, то у функції відладчик починає виконувати всіх операторів крок за кроком. При виборі команди Step Over відладчик виконує функцію як єдине ціле.

Індивідуальні завдання до лабораторної роботи № 4

Розробити програму для обчислення і виводу на екран у вигляді таблиці значень функції F(a,b,c), де a, b, c – дійсні числа на інтервалі від Хнач. до Хкон. з кроком dX.

Варіант 1

$$F = \begin{cases} a*x^2 + b & \text{при } x<0 \text{ і } b \neq 0 \\ (x - a) / (x - з) & \text{при } x>0 \text{ і } b = 0 \\ x / з & \text{у решті випадків} \end{cases}$$

Варіант 2

$$F = \begin{cases} 1 / a*x - b & \text{при } x+5 < 0 \text{ і } c = 0 \\ (x-a) / x & \text{при } x+5 >0 \text{ і } з \neq 0 \\ 10*x / (c-4) & \text{у решті випадків} \end{cases}$$

Варіант 3

$$F = \begin{cases} a*x^2 + b*x + з & \text{при } a <0 \text{ і } з \neq 0 \\ -a / (x-c) & \text{при } a > 0 \text{ і } з = 0 \\ a* (x + з) & \text{у решті випадків} \end{cases}$$

Варіант 4

$$F = \begin{cases} -a*x - з & \text{при } з <0 \text{ і } x \neq 0 \\ (x-a) / -c & \text{при } з > 0 \text{ і } x = 0 \\ b* x / (c-a) & \text{у решті випадків} \end{cases}$$

Варіант 5

$$F = \begin{cases} a - x / (10+b) & \text{при } x <0 \text{ і } b \neq 0 \\ (x-a) / (x-c) & \text{при } x > 0 \text{ і } b = 0 \\ 3*x + 2 / з & \text{у решті випадків} \end{cases}$$

Варіант 6

$$F = \begin{cases} a*x^2 + b^2*x & \text{при } з <0 \text{ і } b \neq 0 \\ (x+a) / (x+c) & \text{при } з > 0 \text{ і } b = 0 \\ x / з & \text{у решті випадків} \end{cases}$$

Варіант7

$$F = \begin{cases} -a*x^2 - b & \text{при } x < 5 \text{ і } з \neq 0 \\ (x+a) / x & \text{при } x > 5 \text{ і } з = 0 \\ -x / з & \text{у решті випадків} \end{cases}$$

Варіант 8

$$F = \begin{cases} -a*x^2 + b & \text{при } з < 0 \text{ і } a \neq 0 \\ (a - x) / c*x & \text{при } з > 0 \text{ і } a = 0 \\ -x / з & \text{у решті випадків} \end{cases}$$

Варіант 9

$$F = \begin{cases} a*x^2 + b*x & \text{при } a < 0 \text{ і } x \neq 0 \\ x - a / (x - з) & \text{при } a > 0 \text{ і } x = 0 \\ -1 + x / з & \text{у решті випадків} \end{cases}$$

Варіант 10

$$F = \begin{cases} a*x^2 + b*x + з & \text{при } x < 3 \text{ і } b \neq 0 \\ (x - a) / (x - з) & \text{при } x > 3 \text{ і } b = 0 \\ x / з & \text{у решті випадків} \end{cases}$$

Варіант 11

$$F = \begin{cases} a*x^2 + b / з & \text{при } x < 1 \text{ і } з \neq 0 \\ (x - a) / (x - з)^2 & \text{при } x > 1{,}5 \text{ і } з = 0 \\ x^2 / c^2 & \text{у решті випадків} \end{cases}$$

Варіант 12

$$F = \begin{cases} a*x2 + b*x + з & \text{при } x < 0{,}6 \text{ і } b + з \neq 0 \\ (x - a) / (x - з) & \text{при } x > 0{,}6 \text{ і } b + з = 0 \\ x / з + x / a & \text{у решті випадків} \end{cases}$$

Варіант 13

$$F = \begin{cases} a*x^2 + b & \text{при } x - 1 < 0 \text{ і } b - x \neq 0 \\ (x - a) / x & \text{при } x - 1 > 0 \text{ і } b + x = 0 \\ x / з & \text{у решті випадків} \end{cases}$$

Варіант 14

$$F = \begin{cases} -a*x^3 - b & \text{при } x + з < 0 \text{ і } a \neq 0 \\ (x - a) / (x - з) & \text{при } x + з > 0 \text{ і } a = 0 \\ x / з + x / з & \text{у решті випадків} \end{cases}$$

Варіант 15

$$F = \begin{cases} -a*x^2 + b & \text{при } x < 0 \text{ і } b \neq 0 \\ x / (x - з) & \text{при } x > 0 \text{ і } b = 0 \\ x / -c & \text{у решті випадків} \end{cases}$$

Варіант 16

$$F = \begin{cases} a*(x + з)^2 - b & \text{при } x = 0 \text{ і } b \neq 0 \\ (x - a) / -c & \text{при } x = 0 \text{ і } b = 0 \\ a + x / -c & \text{у решті випадків} \end{cases}$$

Варіант 17

$$F = \begin{cases} a*x^2 - c*x + b & \text{при } x + 10 < 0 \text{ і } b \neq 0 \\ (x - a) / (x - з) & \text{при } x + 10 > 0 \text{ і } b = 0 \\ -x / (a - з) & \text{у решті випадків} \end{cases}$$

Варіант 18

$$F = \begin{cases} a*x3 + b*x2 & \text{при } x < 0 \text{ і } b \neq 0 \\ (x - a) / (x - з) & \text{при } x > 0 \text{ і } b = 0 \\ (x + 5) / (c* (x - 10)) & \text{у решті випадків} \end{cases}$$

Варіант 19

$$F = \begin{cases} a*(x + 7)^2 - b & \text{при } x < 5 \text{ і } b \neq 0 \\ (x - c*d) / a*x & \text{при } x > 0 \text{ і } b = 0 \\ x / -c & \text{у решті випадків} \end{cases}$$

Варіант 20

$$F = \begin{cases} -(2*x - з) / (c*x - a) & \text{при } x < 0 \text{ і } b \neq 0 \\ (x - a) / (x - з) & \text{при } x > 0 \text{ і } b = 0 \\ (x /c + -c / 2)*x & \text{у решті випадків} \end{cases}$$

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Методичні рекомендації
до виконання лабораторних робіт
з навчальної дисципліни

"АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ"

для студентів напряму підготовки 6.050101
"Комп'ютерні науки"
всіх форм навчання

Укладачі: **Федорченко** Володимир Миколайович
Парфьонов Юрій Едуардович
Щербаков Олександр Всеволодович та ін.

Відповідальний за випуск **Пономаренко В. С.**

Редактор **Бутенко В. О.**

Коректор **Мартовицька-Максимова В. А.**

План 2012 р. Поз. № 267.
Підп. до друку Формат 60 × 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 11,25. Обл.-вид. арк. 14,06. Тираж прим. Зам. №

Видавець і виготівник — видавництво ХНЕУ, 61166, м. Харків, пр. Леніна, 9а

Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи
Дк № 481 від 13.06.2001 р.

Методичні рекомендації
до виконання лабораторних робіт
з навчальної дисципліни

"АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ"

для студентів напряму підготовки 6.050101
"Комп'ютерні науки"
всіх форм навчання