

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторного практикуму
та самостійної роботи
з дисципліни

„ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ“
(частина перша)

для студентів спеціальностей

152 Метрологія та інформаційно-вимірювальна техніка

Галузь знань:

15 Автоматизація та приладобудування
денної й заочної форм навчання

Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни „Обчислювальна техніка та програмування“ (Частина перша) для студентів спеціальності 152 „Метрологія та інформаційно-вимірювальна техніка“ Галузь знань: 15 „Автоматизація та приладобудування” денної й заочної форм навчання /Укл.: В.І.Рева. – Запоріжжя: НУ «Запорізька політехніка», 2019. – 106 с.

Укладачі: В. І. Рева, доц., канд. фіз.-мат. наук

Рецензент: О.В. Василенко, доц., канд. техн. наук

Відповідальний за випуск: А. В. Коротун, доц., канд. фіз.-мат. наук

Затверджено
на засіданні кафедри
мікро- та наноелектроніки

Протокол №2
від “ 30 “ вересня 2019 р.

Рекомендовано до видання
НМК ФРЕТ
Протокол №3
від “ 03 “ жовтня 2019 р.

ЗМІСТ

Вступ	5
Підготовка до проведення лабораторної роботи та оформлення звіту	6
1 Лабораторна робота № 1 «Програмування основних алгоритмічних конструкцій мовами C і C ++»	7
1.1 Короткий опис мов C і C ++	7
1.1.1 Структура програми	7
1.1.2 Прості базові типи	10
1.1.3 Оператор присвоювання	13
1.1.4 Арифметичні і логічні операції	14
1.1.5 Вивід на екран	17
1.1.6 Введення з клавіатури	18
1.1.7 Види алгоритмів	19
1.1.8 Розгалужуються алгоритми	22
1.1.9 Циклічні алгоритми	24
1.1.10 Стандартні бібліотечні функції	33
1.2 Практичне завдання	34
1.2.1 Варіанти завдань	34
2 Лабораторна робота № 2 «Робота з масивами»	40
2.1 Одномірні масиви	40
2.1.1 Дії над елементами масивів	40
2.2 Зв'язок покажчиків і масивів	42
2.3 Заповнення масивів	43
2.4 Пошук елементів, що задовольняють заданій умові	44
2.5 Визначення максимального елемента і його положення в масиві	45
2.6 Упорядкування (сортування) масивів	47
2.7 Видалення елементів з масиву і вставка елементів в масив	55
2.8 Багатовимірні масиви	57
2.9 Масиви вказівників	60
2.10 Практичне завдання	60
2.10.1. Варіанти завдань	61
3 Лабораторна робота № 3 «Робота з текстом, файлами»	67
3.1 Робота з рядками	67
3.1.1 Робота з рядками засобами мови C	67

3.1.2 Масиви рядків	69
3.1.3 Клас string бібліотеки C ++	71
3.2 Робота з файлами	73
3.2.1 Відкриття файлів	73
3.2.2 Текстові та бінарні файли	74
3.2.3 Читання з файлу	75
3.2.4 Запис в файл	76
3.2.5 Закриття файлу	77
3.2.6 Позиціонування покажчика файлу	78
3.2.7 Поняття потоку введення-виведення	78
3.2.8 Функції бібліотеки введення-виведення	79
3.2.9 Класи і об'єкти потоків введення-виведення	79
3.3 Практичне завдання	81
3.3.1 Варіанти завдань	82
Додаток А Приклад оформлення титульної сторінки	85
Додаток Б Підпрограми. Вказівники	86
Додаток В Особливості мультіфайлового програмування ...	95
Додаток Г Елементи об'єктно-орієнтованого програмування	100
Частина 2	

ВСТУП

У лабораторному практикумі розглядаються теоретичні і практичні відомості, необхідні для оволодіння навичками програмування на мовах C і C ++. Практикум складається з 7 лабораторних робіт. В описі кожної лабораторної роботи приведені теоретичні відомості та індивідуальні завдання.

Ні для кого не секрет, що значна частина сучасних радіотехнічних пристроїв побудована на базі мікропроцесорної техніки. Найяскравішими прикладами є мобільні телефони, пейджери, МР3-плеєри, автомобільні сигналізації, пристрої управління інжекторами і т.д. Розробляються численні пристрої, що працюють спільно з міцно ввійшли в повсякденне життя персональними комп'ютерами. Тому сучасному радіоінженеру життєво необхідно володіти навичками програмування.

Серед розробників радіоапаратури загальноприйнятим стандартом при написанні програм є застосування мов C і C ++. Існує багато книг про C / C ++. Однак структура викладу матеріалу в них, як правило, відрізняються від прийнятої при вивченні програмування у ВНЗ. Мабуть, це пов'язано з протиріччям між необхідністю досить суворого викладу матеріалу в лекції

Його зрозумілого роз'яснення (найчастіше з випередженням) на практичних заняттях. Помітний вплив на процес навчання програмування надає специфіка самого мови C ++. З одного боку, це мова, що вивчається поступово. З іншого боку, оволодіння практичними навичками програмування вимагає випереджаючого вивчення, а так-же повернення і переосмислення вже вивченого.

Хочеться звернути особливу увагу читачів на те, що в них вказівках намічена канва вивчення, є посилання на більш об'ємні джерела, сформульовані завдання і намічені шляхи їх вирішення. Використання приведенного матеріалу недостатнім для формування стійкого досвіду програмування. Слід обов'язково розширювати теоретичні знання, використовуючи книги з бібліографічного списку.

ПІДГОТОВКА ДО ПРОВЕДЕННЯ ЛАБОРАТОРНОЇ РОБОТИ ТА ОФОРМЛЕННЯ ЗВІТУ

Лабораторна робота включає самостійне опрацювання теоретичного матеріалу, вивчення документації з програмних засобів, методик проведення розрахунків, програмування і моделювання, обробку й інтерпретацію результатів. При проведенні лабораторного практикуму необхідно:

- підготуватися до експрес-опитування з теоретичного матеріалу, необхідного для виконання роботи (щоб отримати допуск до лабораторної роботи);

- підготувати і оформити план виконання лабораторної роботи;

- після виконання лабораторної роботи провести необхідні розрахунки і оформити звіт, який повинен містити:

- 1) титульний аркуш (див. Додаток А);

- 2) мету роботи;

- 3) опис словами;

- 4) необхідні розрахунки;

- 5) текст розробленої програми;

- 6) основні результати моделювання;

- 7) стислі висновки по роботі, в яких:

- вказати, що досліджувалось;

- вказати методику (чи метод) дослідження;

- вказати особливості алгоритму роботи;

- визначити характер роботи;

- відзначити подібність і відмінність результатів моделювання від теоретичних даних;

- здати звіт з виконаної лабораторної роботи та відповісти на запитання викладача (отримати залік за виконану роботу).

1 ЛАБОРАТОРНА РОБОТА № 1

«ПРОГРАМУВАННЯ ОСНОВНИХ АЛГОРИТМІЧНИХ КОНСТРУКЦІЙ МОВАМИ C і C ++»

Мета роботи: оволодіння практичними навичками роботи з компілятором C / C ++; оволодіння практичними навичками програмування лінійних, розгалужених і циклічних алгоритмічних конструкцій.

1.1 Короткий опис мов C і C ++

Мова програмування C був розроблений на початку сімдесятих років XX століття. До теперішнього часу він зазнав кілька етапів свого розвитку. Мова програмування C ++ об'єднав в собі переваги мови C і можливості об'єктно-орієнтованого програмування. До основних переваг мов C і C ++ слід віднести гнучкість і надійність, переносимість програм з однієї платформи на іншу і з однієї операційної системи на іншу. Мови C і C ++ є мовами високого рівня. Але разом з тим вони мають багаті можливості низько рівневого програмування. Це робить їх дуже гнучкими і придатними як для початківців користувачів, так і для професійних програмістів.

1.1.1 Структура програми

Структура програми на мові C (C ++) досить довільна. Зазвичай спочатку програми розташовуються так звані директиви препроцесору. Кожна директива починається знаком #.

Одна з директив називається include служить для включення в текст програми посилань на бібліотеки мови.

Так, наприклад, рядок : **#include <stdio.h>** включає в текст програми файл stdio.h, в якому розташовані заголовки бібліотечних функцій з бібліотеки stdio. Розширення файлу h свідчить про те, що файл є заголовним. Оскільки stdio є стандартною бібліотекою мови, ім'я файлу stdio.h укладено в кутові дужки. В іншому випадку ім'я файлу необхідно укладати в подвійні лапки.

C-програма складається з підпрограм, які можуть викликати один одного. Кожна підпрограма називається функцією. Зв'язок між функціями організована за допомогою аргументів функцій і повертаються ними значень. Серед всіх функцій одна є головною. Саме з цієї функції, ім'я якої `main`, починається виконання програми. Функція `main` може мати аргументи, за допомогою яких відбувається її взаємодія з операційною системою. Параметри, зазначені в командному рядку при запуску програми, є аргументами функції `main`. С допомогою повертається функція `main` може повідомити операційній системі про результат своєї роботи. Як правило повертається значення, рівне нулю, свідчить про успішне завершення програми. Нульове значення говорить про наявність помилок і може бути проаналізовано операційною системою.

Розглянемо тепер класичний приклад найпростішої програми, яка виводить вітальне повідомлення на екран.

```
1. #include <stdio.h>
2. int main(void)
3. {
4.     printf("Поздравляю!\n");
5.     return 0;
6. }
```

Рядок `int main(void)` описує функцію `main`. Службове слово `void` свідчить про те, що функція `main` не вимагає аргументів. Слово `int` говорить про те, що функція `main` повертає операційній системі ціле значення. Саме це значення вказано після оператора `return` в кінці програми. Тіло функції укладено в фігурні дужки. Таким чином, загальна схема опису будь-якої функції виглядає так:

```
Тип Результату Ім'я Функції (Список Аргументів)
{
Тіло функції;
}
```

Точка в кінці позначає кінець речення і не відноситься до програми.

У розглянутій програмі використана також стандартна функція виведення на екран `printf`. В даному прикладі вона виводить на

екран слово «Поздравляю»! Символи \ n необхідні для того, щоб після виведення вітального повідомлення курсор командного рядка перейшов на наступну терміну. Оператор **return** здійснює вихід з функції і повернення значення, в нашому прикладі нульового.

Ідентифікатори.

Ідентифікатором є послідовність літер, цифр і знаків підкреслення, яка починається з букви або символу підкреслення і не містить пробілів. Ідентифікатори виступають в якості імен будь-яких об'єктів. Символи нижнього і верхнього регістрів в мові C (C++) вважаються різними. Ідентифікатор може мати довільну довжину, проте, значущими є не всі символи. Їх число різне для різних систем програмування, а також може налаштовуватися. Ідентифікатори не повинні збігатися з ключовими словами мови і іменами стандартних бібліотечних функцій.

Коментарі.

Коментарі до програми можна записувати в будь-якому місці програми. Вони відокремлюються символами `/*Це щось коментували*/`. У мові C++ доступна ще одна форма коментаря. Все що написано після знаків `//` і до кінця рядка вважається коментарем.

Змінні.

Програма, як правило, оперує з деякими даними. Для їх зберігання в мовах програмування використовують змінні. Під змінної розуміють іменовану комірку або область пам'яті. Змінні відрізняються типом даних, що зберігаються, ім'ям, значенням. Різні типи даних вимагають різного розміру виділеної пам'яті, а також різної структури цієї пам'яті. Тому перед використанням змінної її потрібно описати (оголосити). При оголошенні необхідно вказати як мінімум її тип і ім'я, наприклад:

1. `int i, j, k;`
2. `float x, y;`

Тут оголошені три змінні цілого типу з іменами `i`, `j`, `k` і дві змінні дійсного типу `x` і `y`. При оголошенні можна вказати значення змінної, наприклад:

1. `int i=1, j=10, k=500;`
2. `float x=3.14, y;`

Таким чином, оголошення змінних в загальному вигляді можна записати як: Тип змінної Список змінних; або Тип змінної Ім'я 1 = знач1, ім'я2 = знач2, ...; .

Важливе значення має питання про місце оголошення змінних. Одні можуть бути оголошені поза будь-яких функцій. Такі змінні називаються глобальними. Вони доступні для використання в будь-якому місці програми, будь-якої функції. Інші змінні можуть оголошуватися всередині будь-якого блоку, наприклад, всередині функції. Такі змінні називають локальними. Їх можна використовувати тільки всередині цього блоку. Ніяка функція не може використовувати змінні, оголошені всередині іншої функції. Тому в різних блоках можна використовувати змінні з однаковими іменами, хоча за змістом це будуть різні змінні.

1.1.2 Прості базові типи

Типи даних всіх мов програмування можна розділити на прості та структуровані. Мінлива простого типу зберігає одне значення. З мовою C (C++) це цілі і речові числа, символи та покажчики. Змінні структурованих типів зберігають багато значень простих або теж структурованих типів. До таких типів належать масиви, структури, об'єкти і т.д. У мові введені 5 простих типів, які називають базовими.

1. Змінна типу `char` займає в пам'яті 1 байт. Її значеннями можуть бути символи кодової таблиці.
2. Змінна типу `int` використовується для зберігання цілих чисел і має довжину 2 байта.
3. Для використання речових даних передбачений тип `float` довжиною 4 байта.
4. Тип `double` відповідає речовим числам подвійної точності. Мінлива такого типу займає 8 байт пам'яті.
5. Слово `void` використовується для декларування об'єкта, що не має значення, наприклад, при описі функції, не повертає ніякого значення або не має аргументів.

Перераховані базові типи можуть бути використані спільно з модифікаторами, які ставляться перед назвою типу: `signed` знаковий, `unsigned` беззнаковий, `long` довгий, `short` короткий.

Цілі типи.

Змінні одного з цілих типів призначені для зберігання цілих чисел зі знаком або без знаку (тільки позитивні і нуль). Базовим цілим типом є тип `int`, для якого модифікатори `signed` і `short` є модифікаторами за замовчуванням. Таким чином, для оголошення змінної `M` цілого типу рівносильні наступні опису:

```
int M;
short int M;
signed int M;
signed short int M;
```

Змінна типу `int` займає 2 байта пам'яті і може приймати значення від -32768 до 32767. Беззнакові цілі змінні `unsigned int` також займають 2 байти пам'яті, але можуть приймати значення від 0 до 65535. Очевидно, що в програмі може знадобитися зберігати дані, які не вкладаються в діапазон значень, що надається цілим типом. У цьому випадку доцільно використовувати довгий цілий тип `long int`. Змінні цього типу займають по 4 байти пам'яті і можуть приймати значення від -2147483648 до 2147483647. Беззнаковий довгий цілий тип `unsigned long int` передбачає для зберігання даних 4 байта і діапазон значень від 0 до 4294967295. Характеристики перерахованих цілих типів приведені в табл. 1.1.

Таблиця 1.1 – Цілі типи

Назва	Ідентифікатор	Діапазон	Розмір пам'яті
Знаковий цілий	<code>int</code> <code>signed int</code> <code>short int</code> <code>signed short int</code>	-32768.. ..32768	2 байта
Беззнаковий цілий	<code>unsigned int</code> <code>unsigned short int</code>	0..65535	2 байта
довгий знаковий цілий	<code>long int</code> <code>signed long int</code>	-2147483648 .. 2147483647	4 байта
Беззнаковий довгий цілий	<code>unsigned long int</code>	0..4294967295	4 байта

Типи з плаваючою комою.

У групу типів з плаваючою комою входять три типи: `float`, `double` і `long double`. Вони відрізняються кількістю виділеної пам'яті, а також діапазоном можливих значень. Характеристики речових типів приведені в табл. 1.2.

Таблиця 1.2 – Типи з плаваючою комою

Назва	Ідентифікатор	Діапазон значень по модулю	Розмір пам'яті
Речове одинарної точності	<code>float</code>	от $3:4 \cdot 10^{38}$ до $3:4 \cdot 10^{38}$	4 байта
Речове подвійної точності	<code>double</code>	от $1:7 \cdot 10^{308}$ до $1:7 \cdot 10^{308}$	8 байт
Речове підвищеної точності	<code>long double</code>	от $3:4 \cdot 10^{4932}$ до $1:1 \cdot 10^{4932}$	10 байт

Символьний тип `char`.

Змінні і константи цього типу можуть набувати значень з безлічі символів кодової таблиці і займають в пам'яті 1 байт. Символ, укладений в апострофи, позначає константу символьного типу, наприклад: `'5'`, `'d'`. Однак в пам'яті зберігається саме код символу з кодовою таблиці. Тому символи також можна інтерпретувати як цілі числа і застосовувати до них модифікатори `signed` і `unsigned`. За замовчуванням для типу `char` використовується модифікатор `signed`. Отже значення змінних символьного типу інтерпретуються як цілі числа зі знаком від -128 до 127, що ілюструє табл. 1.3.

Таблиця 1.3 – Символьні типи

Ідентифікатор	Діапазон значень	Розмір пам'яті
<code>char</code> или <code>signed char</code>	-128..127	1 байт
<code>unsigned char</code>	0..255	1 байт

Символи з кодами від 0 до 31 є керуючими символами (див. Додаток), вони не відповідають жодному друкованому символу, їх не можна ввести з клавіатури. Для можливості використання в мові C (C++) передбачені керуючі константи, деякі з яких перераховані в табл. 1.4

Таблиця 1.4 – Керуючі константи

Послідовність	Значення
<code>\b</code>	BS, забой
<code>\f</code>	Нова сторінка, переклад сторінки
<code>\n</code>	Новий рядок, новий рядок
<code>\r</code>	повернення каретки
<code>\t</code>	горизонтальна табуляція
<code>\v</code>	вертикальна табуляція
<code>\"</code>	лапки
<code>\'</code>	апостроф
<code>\\</code>	зворотний слеш
<code>\0</code>	Нульовий символ, нульовий байт
<code>\a</code>	Звуковий сигнал

1.1.3 Оператор присвоювання

Для мови C++ прояви операції присвоювання дуже різноманітні. Проста форма операції присвоювання задається знаком `=`, наприклад, `pi = 3.14;`. Крім основної в мові C++ існують спеціальні форми оператора присвоювання, приклади яких показані в табл. 1.5.

Таблиця 1.5 – Форми оператора присвоювання

Звичайний запис	Більш коротко на C++
$A = A * B$	$A *= B$
$A = A / (B + 4)$	$A /= B + 4$
$A = A \% B$	$A \% = B$
$A = A + B$	$A += B$
$A = A - B$	$A -= B$

1.1.4 Арифметичні і логічні операції

У програмуванні для визначення дій служать вираження (аналог математичних формул). Вирази складаються з операцій і операндів. Операції поділяються на унарні (один операнд) і бінарні (два операнда). У табл. 1.6 перераховані унарні і бінарні операції, використовувані в C++ частіше за інших.

Таблиця 1.6 – Операції C++

Операція	Короткий опис
Унарні операції	
::	доступ до області видимості
.	вибір
->	вибір
[]	індексація
()	виклик функції
<тип>()	конструювання
++	постфіксний інкремент
—	постфіксний декремент
typeid	ідентифікація типу

Продовження таблиці 1.6

dynamic_cast	перетворення типу з перевіркою на етапі виконання
static_cast	перетворення типу з перевіркою на етапі компіляції
reinterpret_cast	перетворення типу без перевірки
const_cast	константне перетворення типу
sizeof	розмір об'єкта або типу
—	префіксний декремент
++	префіксний інкремент
~	поразрядное заперечення
!	логічне заперечення
—	арифметичне заперечення (унарний мінус)
+	унарний плюс
&	взяття адреси
*	разадресація
new	виділення пам'яті
delete	звільнення пам'яті
(<тип>)	перетворення типу
.*	вибір
->*	вибір

Продовження таблиці 1.6

Бінарні і тринарні операції

*	множення
/	поділ
%	залишок від ділення
+	складання
—	віднімання
<<	зсув вліво
>>	зсув вправо
<	менше
<=	менше або дорівнює
>	більше
>=	більше чи рівно
==	одно
!=	не дорівнює
&	поразрядное кон'юнкція (І)
^	поразрядное виключає АБО
 	поразрядное диз'юнкція (АБО)
&&	логічне І
 	логічне АБО

Продовження таблиці 1.6

? :	умовна операція (тернарного)
=	присвоювання
*=	множення з присласненням
/=	розподіл з присласненням
%=	залишок відділення з присласненням
+=	складання з присласненням
-=	віднімання з присласненням
<<=	зсув вліво з присласненням
>>=	зсув вправо з присласненням
&=	поразрядное І з присласненням
 =	поразрядное АБО з присласненням
^=	поразрядное виключає АБО з присласненням
throw	виняток
,	послідовне обчислення

1.1.5 Вивід на екран

Спілкування з зовнішнім світом програма на мові С реалізує за допомогою бібліотеки функцій вводу-виводу `stdio`. Для її використання необхідно включити в програму заголовки бібліотеки:

1. `#include <stdio.h>`
2. Одним із способів виведення на екран є використання функції `printf`:
3. `printf ("Керуюча рядок", Arg1, Arg2, ...);`

Керуюча рядок може містити компоненти трьох типів:

- ❖ звичайні символи, які просто виводяться на екран;
- ❖ керуючі константи з табл. 1.5;
- ❖ специфікації перетворення.

Кожна специфікація перетворення служить для виведення на екран чергового даного зі списку аргументів Arg1, Arg2, ... Починається специфікація знаком%, закінчується символом перетворення. У наступному прикладі:

printf ("Значення числа Пі одно% f \ n", pi)

символом перетворення є буква f. Отже функція printf виведе на екран замість% f значення змінної pi, яка знаходиться в списку аргументів.

Кожному аргументу повинна відповідати своя специфікація перетворення:

- % D - десяткове ціле число;
- % F - дійсне число;
- % C - символ;
- % S - рядок.

У мові C ++ на основі об'єктно-орієнтованого способу програмування реалізований так званий механізм потокових об'єктів. В результаті підключення до програми бібліотеки функцій для роботи з потоковими об'єктами

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

стає доступний об'єкт вихідного потоку cout, пов'язаний з екраном. Тому для виведення на екран досить виведений об'єкт направити в потік. Ось кілька прикладів:

1. `cout << "Поздравляем !!! \ n";`
2. `cout << "Значення Пі одно" << pi << "\ n";`
`//.`

Тут операція приміщення в потік << виконується ланцюжком зліва направо.

1.1.6 Введення з клавіатури

Для введення значень змінних з клавіатури служить функція `scanf`:

```
scanf ("Керуюча рядок", Arg1, Arg2, ...);
```

Функція `scanf` працює аналогічно функції `printf`, але керуючий рядок повинний складатися тільки з специфікацій перетворення. В якості аргументів вказують імена змінних, яким будуть привласнені введені данні, з символом `&` спочатку, наприклад,

```
scanf ("% d% f", & x, & y); //.
```

У мові C ++ введення з клавіатури можна здійснити допомогою читання з потокового об'єкта `cin`, пов'язаного з клавіатурою, наприклад,

```
cout << "Введіть значення x"; cin >> x;  
cout << "\ n"; //.
```

1.1.7 Види алгоритмів

За структурою всі програми можна розділити на 3 основні групи: лінійні, розгалужуються і циклічні.

Лінійним називається обчислювальний процес (алгоритм), в якому дії виконуються в лінійній послідовності, одне за іншим, і кожна дія виконується один раз. Зазвичай такі алгоритми складаються з команд присвоювання.

Виконання розгалужуючого алгоритму відбувається через одну з таких заздалегідь запланованих гілок в залежності від виконання або невиконання деякої умови. Наприклад, алгоритм вирішення квадратного рівняння залежить від знака дискримінанта.

У циклічному алгоритмі одні і ті ж дії виконуються багаторазово. Такий багаторазово повторюється ділянку алгоритму називається циклом.

Приклад.

Розглянемо процес створення найпростішої програми з лінійним алгоритмом на прикладі наступної задачі.

Напишіть програму обчислення доходу по вкладу. Процентна ставка (у відсотках річних) і час зберігання (в днях) задаються під час роботи програми.

Наведемо текст програми повністю, а потім обговоримо її зміст.

```
1. #include <iostream.h>  
2. float sum, // сума вкладу
```

```

3. srok, // термін вкладу
4. stavka, // процентна ставка
5. dohod; // дохід за вкладом
6. int main (void)
7. {
8. cout << "Обчислення доходу за вкладом \ n";
9. cout << "Введіть вихідні дані: \ n";
10.    cout << "Величина вкладу (грн) ->";
11.    cin >> sum;
12.    cout << "\ n";
13.    cout << "Термін вкладу (днів) ->";
14.    cin >> srok;
15.    cout << "\ n";
16.    cout << "Процентна ставка ->";
17.    cin >> stavka;
18.    cout << "\ n";
19.    dohod = (sum * stavka / 100) * srok /
    365;
20.    sum = sum + dohod;
21.    cout << "\ n";
22.    cout << "----- \
    n"; cout << "Дохід:" << dohod << "грн \ n";
23.    cout << "Кінцева сума:" << sum << "грн
    \ n"; } //

```

кінець програми.

Спочатку програми рядком `#include <iostream.h>` включається заголовки бібліотеки **iostream** для використання потокових об'єктів виведення на екран і введення з клавіатури. Далі оголошуються змінні **sum**, **srok**, **stavka**, **dohod** дійсного типу **float**. Головна і єдина функція **main**, з якої починається виконання програми описана як `int main (void)`. Це означає, що функція не залежить ні від яких аргументів і може передати операційній системі

цілочисельний результат своєї роботи. `cout << "Обчислення доходу за вкладом \ n";`

`cout << "Введіть вихідні дані: \ n";`

виводять на екран запрошень інформацію. Інформація, що виводиться рядок закінчується посднанням `\ n`. Це призводить до того, що після виведення на екран курсор переходить на наступний рядок, де і буде розташовуватися наступний висновок на екран або введення даних.

Вихідні дані користувач буде вводити з клавіатури. Для зручності спочатку виводиться підказка

`cout << "Величина вkladу (грн) ->";`

потім команда: `cin >> sum;` - здійснює введення з клавіатури значення початкової суми, покладеної в банк. Введене значення присвоюється змінної `sum`. Рядок: `cout << "\n";` - виводить на екран недрукований символ `\ n`, в результаті курсор переводиться на наступний рядок. Аналогічно виконується введення терміну вkladу в днях

`cout << "Термін вkladу (днів) ->";`

`cin >> srok;`

`cout << "\ n";`

а також величини процентної ставки:

`cout << "Процентна ставка ->";`

`cin >> stavka;`

`cout << "\ n"; //.`

Введені значення присвоюються змінним `srok` і `stavka` відповідно. Розрахунок доходу за вкладом виконується за формулою:

`dohod = (sum * stavka/100) * srok/365;`

тому такі два рядки програми є оператори присвоювання, перший з яких обчислює дохід, а другий змінює значення суми вkladу:

1. `dohod=(sum*stavka/100)*srok/365;`

2. `sum=sum+dohod; //.`

Результати обчислень виводяться на екран:

1. `cout << "\ n";`

```

2. cout << "----- \ n";
3. cout << "Дохід:" << dohod << "грн \ n";
4. cout << "Кінцева сума:" << sum << "грн \ n";
   // .

```

1.1.8 Розгалужуються алгоритми.

Для запису алгоритмів, що розгалужуються на мові С (С++) передбачені два оператора розгалуження `if` і `switch`.

Оператор `if` здійснює розгалуження за двома напрямками і в загальному вигляді виглядає так:

```
if (Умова) Оператор1; else Оператор2; // .
```

При виконанні оператора `if` спочатку виконується перевірка умови. Якщо воно істинне, то виконується Оператор 1, інакше Оператор 2. Блок-схема, відповідна оператору `if` зображена на рис. 1.1. В якості умови може використовуватися не тільки логічне вираження. Будь-яке вираження, що приймає нульове значення, вважається дійсним, нульове значення хибним.

Допускається скорочений варіант оператора `if`:

```
if (Умова) Оператор1//.
```

Тоді при істинності умови виконується Оператор 1, в іншому випадку виконується наступний після `if` оператор. Блок-схема відповідна скороченій формі оператора `if` зображена на рис. 1.2.

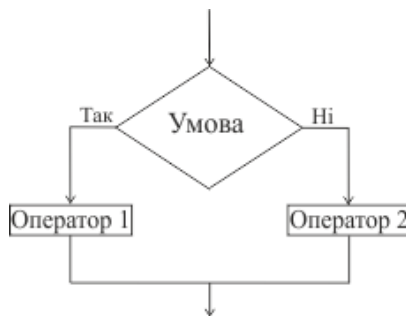


Рисунок 1.1 – Блок-схема розгалуженого алгоритму.

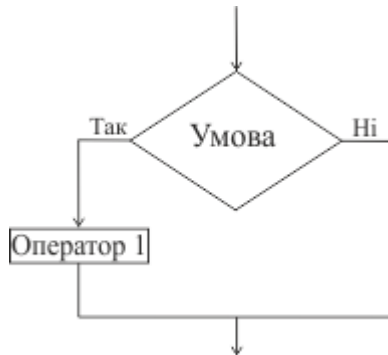


Рисунок 1.2 – Блок-схема скороченого розгалуження

Якщо за змістом завдання замість першого або другого оператора необхідна послідовність з декількох операторів, то їх укладають в фігурні дужки {...}:

```

{
Оператор11;
Оператор12;
...
Оператор1N:
};    //.
```

Отримана конструкція називається складовим оператором.

Приклад.

Дано довжини трьох сторін трикутника. Якщо такий трикутник існує, обчислити його площу.

Програма:

```

1. #include <iostream.h>
2. #include <math.h>
3. double a,b,c,p,s;
4. int main(void)
5. {
6.   cout << "Введіть довжини сторін
   трикутника\n"; cin >> a >> b >> c;
7.   if ((a+b<c) || (a+c<b) || (b+c<a))
```

```

8. cout << "Такого трикутника не
   существует\n";
9. else
10. {
11.   p=(a+b+c)/2;
12.   s=sqrt(p*(p-a)*(p-b)*(p-c));
13.   cout << "Площадь трикутника равна" <<
      s << "\n"; };
14.   }//.

```

Тут використана логічна операція АБО, позначена двома вертикальними лініями, а також стандартна функція обчислення квадратного кореня `sqrt` з бібліотеки `math`

1.1.9 Циклічні алгоритми

Будь-який циклічний алгоритм складається з наступних компонентів: підготовка циклу, перевірка умови продовження або закінчення циклу, модифікація умови і тіло циклу (багаторазово виконуються оператори). Залежно від взаємного розташування компонентів, усі цикли прийнято ділити на цикли з передумовою і умовою поста. Якщо перевірка умови знаходиться перед тілом циклу, то такий цикл відноситься до циклів з передумовою. Його блок-схема зображена на рис. 1.3. В іншому випадку цикл є циклом з умовою поста. Його блок-схема показана на рис. 1.4.

Для організації циклів в мові C (C++) передбачені три оператора.

Оператор з передумовою `while`

`while` (Вираз) Оператор; `//`.

Вираз в дужках це умова продовження циклу, Оператор простий або складовий являє собою тіло циклу. Як умова продовження може використовуватися не тільки логічне вираження. Будь-яке вираження, що приймає нульове значення вважається дійсним, нульове значення хибним. Однак в тілі необхідно передбачити модифікацію умови для того, щоб цикл міг на якомусь етапі закінчитися. Перевірка умови продовження циклу здійснюється перед виконанням тіла. Тому

оператор `while` називається оператором циклу з передумовою. Його тіло може ні разу не виконатися.

Приклад.

Дано 100 дійсних чисел, що вводяться з клавіатури. Знайти різницю між максимальним і мінімальним з них.

Пошук максимального і мінімального значення з багатьох введених даних можна здійснити в такий спосіб. Скористаємося змінними **max** і **min** того ж типу, що й дані, що вводяться. Перше введене число є і максимальним і мінімальним одночасно.

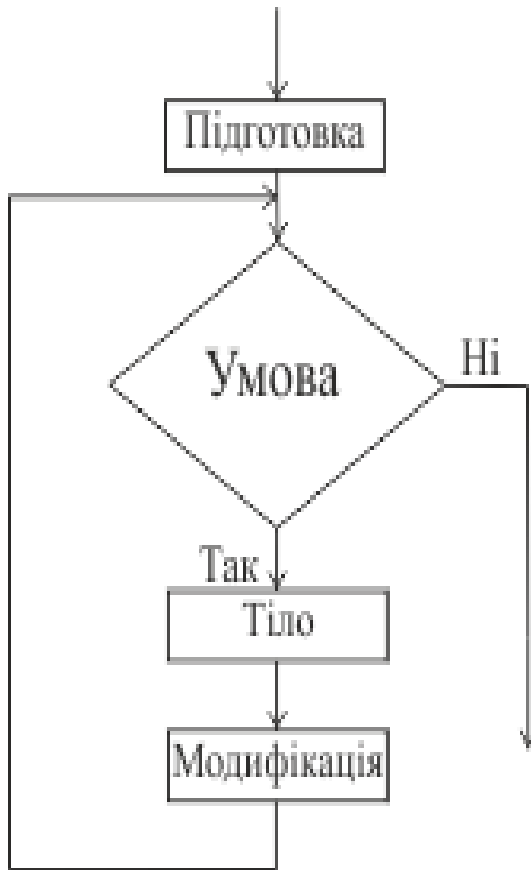


Рисунок 1.3 – Блок-схема циклу з передумовою



Рисунок 1.4 – Блок-схема циклу з умовою поста

Тому дамо змінним **max** і **min** однакове значення, рівне першому введеному числу. Далі багаторазово будемо виконувати введення чергового числа, порівняння його з **max** і **min**. Для цього використовуємо оператор циклу. Щоб перерахувати кількість введених чисел скористаємося змінної **j** цілого типу. Початкове значення змінної **j** виберемо рівним 1. В тілі циклу значення змінної **j** будемо збільшувати на 1, а умовою продовження циклу виберемо нерівність **j < 100**. Нижче наведено текст програми, відповідний описаним алгоритмом, а на рис. 1.5 показана його блок-схема.

```

1. #include <iostream.h>
2. float x,min,max;
3. int j;
4. int main (void)
5. {
6. cout << "Вводите числа!";
7. cin >> x;

```

```

8. min=x; max=x;
9. j=1;
10. while (j<100)
11. {
12.   cin >> x;
13.   if (x<min) min=x;
14.   if (x>max) max=x;
15.   j++; // Это эквивалентно i=i+1 };
16.   cout << "Искомая разность равна" <<
      max-min << "\n";}

```

2) Оператор з умовою поста `do while`

```

1. do
2. {Тіло циклу}
3. while (Вираз);  //.

```

Вираз це умова продовження циклу. Тіло циклу може складатися з великої кількості операторів, серед яких необхідно передбачити модифікацію умови для того, щоб цикл міг на якомусь етапі закінчитися. Перевірка умови продовження циклу здійснюється після виконання тіла. Тому оператор циклу `do ... while` називається оператором циклу з умовою поста. Тіло такого циклу хоча б один раз виконається.

Приклад.

Дано 100 дійсних чисел, що вводяться з клавіатури. Знайти різницю між максимальним і мінімальним з них.

Наведемо текст програми, заснований на попередньому прикладі.

```

1. #include <iostream.h>
2. float x,min,max;
3. int j;
4. int main (void)
5. {
6.   cout << "Вводите числа!";
7.   cin >> x;

```

```

8. min=x; max=x;
9. j=1;
10.    do
11.        {cin >> x;
12.        if (x<min) min=x;
13.        if (x>max) max=x;
14.        j++;}
15.    while (j<100);
16.    cout << "Искомая разность равна" <<
        max-min << "\n";
17.    }

```

Блок-схема рішення задачі з використанням оператора циклу з умовою поста зображена на рис. 12

Оператор циклу з параметром застосовується для організації циклів з відомим кількістю повторень і відноситься до циклів з передумовою. Формат оператора наступний:

For (Вираз1; Вираженіє2; вираз3) Тіло; //.

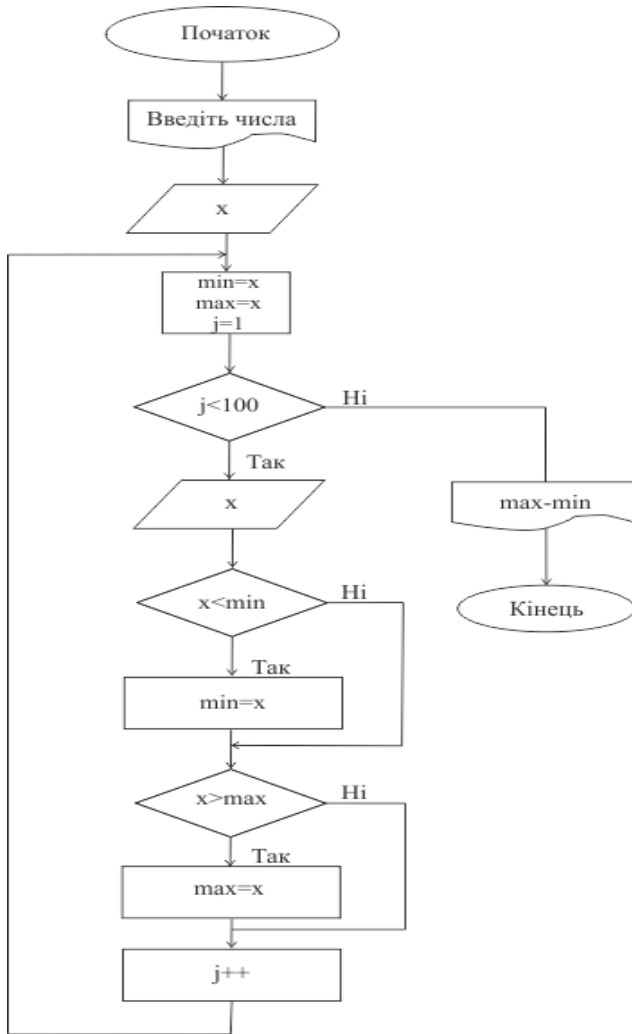


Рисунок 1.5 – Блок-схема рішення задачі при використанні оператора циклу з передумовою

У виразах 1-3 повинні фігурувати змінна циклу. Її значення змінюється в ході виконання циклу і по ньому встановлюється необхідність повторення циклу або виходу з нього. Вираз 1 служить для присвоєння початкового значення змінної циклу. За допомогою

Вирази 2 відбувається перевірка умови продовження циклу, а Вираз 3 змінюється значення змінної циклу, наприклад,

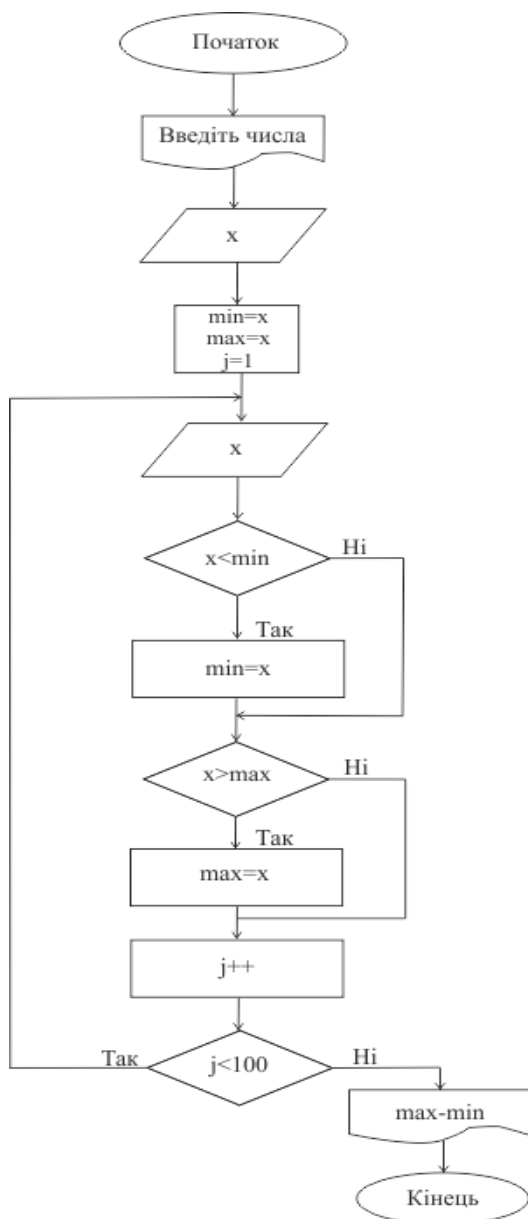
```
for (i = 1; i <= 100; i ++) { ... } //.
```

Приклад.

Дано 100 дійсних чисел, що вводяться з клавіатури. Знайти різницю між максимальним і мінімальним з них.

Знову модифікуємо текст програми попереднього прикладу.

```
1. #include <iostream.h>
2. float x,min,max;
3. int j;
4. int main (void)
5. {
6.     cout << "Вводите числа!";
7.     cin >> x;
8.     min=x; max=x;
9.     j=1;
10.    for (j=2; j<=100; j++)
11.    {
12.        cin >> x;
13.        if (x<min) min=x;
14.        if (x>max) max=x;
15.    };
16.    cout << "Искомая разность равна" <<
        max-min << "\n";
17.    } //.
```



Риснок 1.6 – Блок-схема рішення задачі при використанні оператора циклу з умовою поста

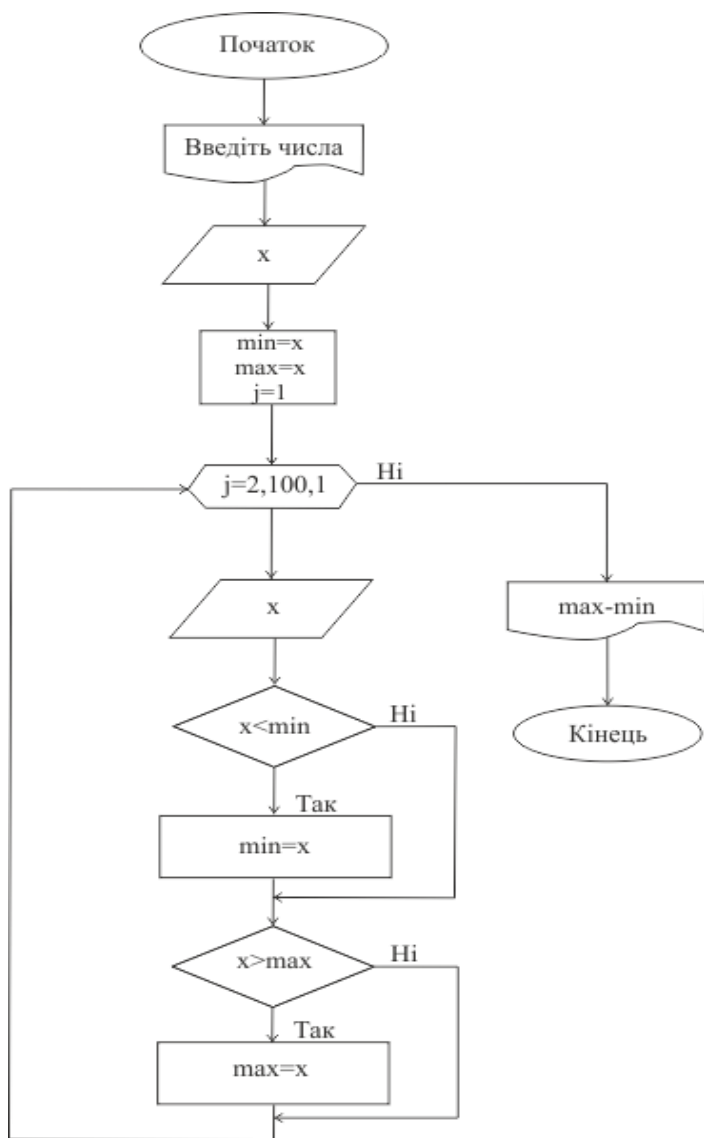


Рисунок 1.7 – Блок-схема рішення задачі при використанні оператора циклу з параметром

Оскільки для завдання початкових значень змінним $\min$ і $\max$ перше число вводиться поза циклом, змінна циклу j приймає значення

від 2 до 100. В середині тіла циклу модифікації змінної j не передбачено. Блок-схема рішення задачі з використанням оператора циклу з параметром зображена на рис. 1.7.

1.1.10 Стандартні бібліотечні функції

Перерахуємо окремим пунктом найуживаніші бібліотечні функції (див. Табл. 1.7). Більш повну інформацію про бібліотечні функції можна знайти в додатку книги і в книзі.

Таблиця 1.7 – Бібліотечні функції

Ім'я функції	Сенс
Функції бібліотеки math	
cos	арккосинус
asin	арксинус
atan	арктангенс
cos	косинус
sin	синус
tan	тангенс
cosh	гіперболічний косинус
sinh	гіперболічний синус
tanh	гіперболічний тангенс
exp	експонента
log	натуральний логарифм
log10	десятковий логарифм
pow	Піднесення до степеня $\text{pow}(x, y) = x^y$
sqrt	Квадратний корінь
ceil	Округлення до більшого
fabs	Модуль дійсного числа
floor	Округлення до меншого
Функції бібліотеки stdlib	
atof	Перетворення рядка в число типу float
atoi	Перетворення рядка в число типу int
atol	Перетворення рядка в число типу long int
rand	Генерація випадкового числа
abs	Модуль цілого числа
labs	Модуль довгого цілого

1.2 Практичне завдання

При виконанні практичного завдання дотримуйтесь наступної послідовності дій:

- 1) Обов'язково вивчіть опис, пропонуване вище.
- 2) Розберіть приклади, що наводяться в описі.
- 3) Осмисліть завдання, запропоновану у Вашому варіанті.
- 4) Складіть словесний алгоритм вирішення задачі.
- 5) Уявіть алгоритм рішення у вигляді блок-схеми.
- 6) Переведіть алгоритм на мову програмування.

1.2.1 Варіанти завдань

Варіант 1.

Для двох натуральних чисел P і $Q < 1000000$ напишіть програму, яка визначає, чи є вони взаємно простими. Два числа називаються взаємно простими, якщо вони не мають спільних дільників більших за 1.

Варіант 2.

У зверненні є монети 1-, 5-, 10-, 50-копійчаного гідності. Напишіть програму, яка визначить задачу найменшою кількістю монет.

Варіант 3.

Натуральне число є досконалим, якщо воно дорівнює сумі всіх своїх дільників, включаючи 1. Напишіть програму, яка виводить на екран вчинені числа, менше 1000.

Варіант 4.

Натуральне число є простим, якщо воно не має дільників, крім самого себе і одиниці. Напишіть програму, яка виводить на екран прості числа, менше 1000.

Варіант 5.

Вихідними даними є координати точки $M(x, y)$. Складіть програму, яка визначає, чи належить точка M області перетину кіл, зображеної на рис. 1.8.

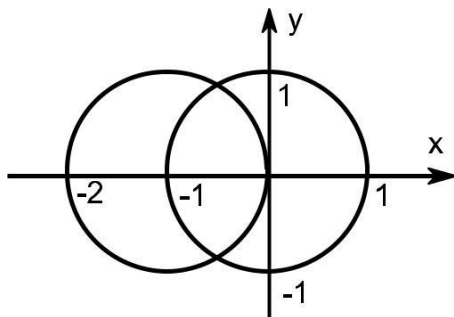


Рисунок 1.8 – Малюнок до варіанту 5

Варіант 6.

Вихідними даними є координати точки $M(x, y)$. Складіть програму, яка визначає, чи належить точка M фігурі, зображеної на рис. 15.

Варіант 7.

Складіть програму, яка обчислює значення виразу:

$$S = \sqrt{3 + \sqrt{6 + \sqrt{9 + \dots + \sqrt{96 + \sqrt{99}}}}}$$

Варіант 8.

Складіть програму, яка обчислює значення виразу:

$$S = \cos(1 + \cos(2 + \cos(3 + \dots + \cos(39 + \cos(40)) \dots)))$$

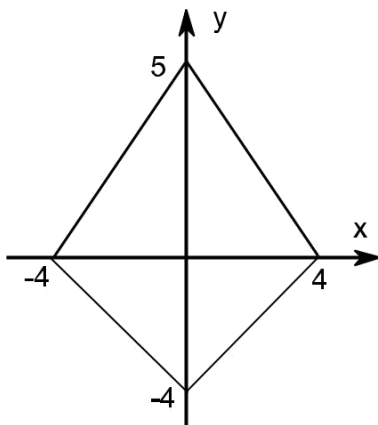


Рисунок 1.9 – Малюнок до варіанту 6

Варіант 9.

Напишіть програму, яка обчислює кількість точок з цілочисельними координатами, які перебувають в колі радіуса $R > 0$ з центром в точці $A(x, y)$.

Варіант 10.

З клавіатури вводиться десяткове число M і підстава системи числення від 2 до 9. Складіть програму, яка переводить число M в зазначену систему числення.

Варіант 11.

Складіть програму розрахунку суми $1 + 2 + 3 + 4 + \dots + N$. Отриманий результат порівняйте з контрольним значенням, обчисленим за формулою $N(N + 1) / 2$. Кількість доданків N вводиться з клавіатури.

Варіант 12.

Складіть програму розрахунку суми $1 + 3 + 5 + 7 + \dots + (2N - 1)$. Отриманий результат порівняйте з контрольним значенням, обчисленим за формулою N^2 . Кількість доданків N вводиться з клавіатури.

Варіант 13.

Складіть програму розрахунку суми $2 + 4 + 6 + 8 + \dots + 2N$. Отриманий результат порівняйте з контрольним значенням, обчисленим за формулою $N(N + 1)$. Кількість доданків N вводиться з клавіатури.

Варіант 14.

Складіть програму розрахунку суми $1^2 + 2^2 + 3^2 + 4^2 + \dots + N^2$. Отриманий результат порівняйте з контрольним значенням, обчисленим за формулою $N(N + 1)(2N + 1) / 6$. Кількість доданків N вводиться з клавіатури.

Варіант 15.

Складіть програму розрахунку суми $1^2 + 3^2 + 5^2 + 7^2 + \dots + (2N - 1)^2$. Отриманий результат порівняйте з контрольним значенням, обчисленим за формулою $N(4N^2 + 1) / 3$. Кількість доданків N вводиться з клавіатури.

Варіант 16.

Складіть програму розрахунку суми $1^3 + 2^3 + 3^3 + 4^3 + \dots + N^3$. Отриманий результат порівняйте з контрольним значенням, обчисленим за формулою $N^2 (N + 1)^2 / 4$. Кількість доданків N вводиться з клавіатури.

Варіант 17.

Складіть програму розрахунку суми $1^3 + 3^3 + 5^3 + 7^3 + \dots + (2N - 1)^3$. Отриманий результат порівняйте з контрольним значенням, обчисленим за формулою $N^2(2N^2 - 1)$. Кількість доданків N вводиться з клавіатури.

Варіант 18.

Складіть програму розрахунку суми $1^4 + 2^4 + 3^4 + 4^4 + \dots + N^4$. Отриманий результат порівняйте з контрольним значенням, обчисленим за формулою $(N^2 + N) (2N + 1) (3N^2 + 3N - 1) / 30$. Кількість доданків N вводиться з клавіатури.

Варіант 19.

Складіть програму наближеного розрахунку нескінченної суми

$$\sum_{i=1}^{\infty} i^{-2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots$$

Підсумовування необхідно проводити, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-4}$, що характеризує точність. Результат підсумовування порівняйте з точним значенням $\pi^2 / 6$. Обчисліть похибку і зіставте її з точністю.

Варіант 20.

Складіть програму наближеного розрахунку нескінченної суми

$$\sum_{i=1}^{\infty} (-1)^{i-1} i^{-2} = 1 - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \dots$$

Підсумовування необхідно проводити, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-5}$, що характеризує точність. Результат підсумовування порівняйте з точним значенням $\pi^2/12$. Обчисліть похибку і зіставте її з точністю.

Варіант 21.

Складіть програму наближеного розрахунку нескінченної суми

$$\sum_{i=1}^{\infty} (2i-1)^{-2} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \dots$$

Підсумовування необхідно проводити, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-4}$, що характеризує точність. Результат підсумовування порівняйте з точним значенням $\pi^2/8$. Обчисліть похибку і зіставте її з точністю.

Варіант 22.

Складіть програму наближеного розрахунку нескінченної суми

$$\sum_{i=1}^{\infty} \frac{(-1)^{i+1}}{2i-1} = 1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \dots$$

Підсумовування необхідно проводити, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-4}$, що характеризує точність. Результат підсумовування порівняйте з точним значенням $\pi/4$. Обчисліть похибку і зіставте її з точністю.

Варіант 23.

Складіть програму наближеного розрахунку $\sin(x)$ у вигляді нескінченної суми

$$\sin(x) = \sum_{i=1}^{\infty} (-1)^{i-1} \frac{x^{2i-1}}{(2i-1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

для значення x , запитуваної з клавіатури. Підсумовування необхідно проводити до тих пір, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-4}$.

Варіант 24.

Складіть програму наближеного розрахунку e^x у вигляді нескінченної суми

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

для значення x , запитуваної з клавіатури. Підсумовування необхідно проводити до тих пір, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-3}$.

Варіант 25.

Складіть програму наближеного розрахунку $\cos(x)$ у вигляді нескінченної суми

$$\cos(x) = \sum_{i=1}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots$$

для значення x , запитуваної з клавіатури. Підсумовування необхідно проводити до тих пір, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-4}$.

Варіант 26.

Складіть програму наближеного розрахунку $\operatorname{sh}(x)$ у вигляді нескінченної суми

$$\operatorname{sh}(x) = \sum_{i=1}^{\infty} \frac{x^{2i-1}}{(2i-1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots$$

для значення x , запитуваної з клавіатури. Підсумовування необхідно проводити до тих пір, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-3}$.

Варіант 27.

Складіть програму наближеного розрахунку $\operatorname{ch}(x)$ у вигляді нескінченної суми

$$\operatorname{ch}(x) = \sum_{i=0}^{\infty} \frac{x^{2i}}{(2i)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \dots$$

для значення x , запитуваної з клавіатури. Підсумовування необхідно проводити до тих пір, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-3}$.

Варіант 28.

Складіть програму наближеного розрахунку $\sin(x)/x$ в вигляді нескінченної суми

$$\frac{\sin(x)}{x} = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i+1)!} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \dots$$

для значення x , запитуваної з клавіатури. Підсумовування необхідно проводити до тих пір, поки модуль чергового доданка не стане менше заздалегідь заданого маленького числа $\epsilon = 10^{-4}$.

2 ЛАБОРАТОРНА РОБОТА № 2 «РОБОТА З МАСИВАМИ»

Мета роботи: оволодіння практичними навичками роботи з масивами в C ++; освоєння методів сортування масивів.

2.1 Одномірні масиви

Масив це формальне об'єднання декількох однотипних змінних (числових, символьних, строкових), що розглядається як єдине ціле. Описати масив можна або при описі змінних, або з використанням опису нового типу. При описі масиву необхідно вказати: ім'я масиву, тип його елементів, кількість елементів в масиві.

Опис масиву в програмі виглядає так:

Тип_Елементов_Массива Ім'я_Массива [Кількість];
наприклад,

```
int a [100], b [100], c [100], d [100];
```

Тут визначено 4 масиву з іменами a, b, c і d. Всі вони складаються з елементів цілого типу, пронумерованих цілими числами від 0 до 99. Елементи масиву завжди нумеруються цілими числами, починаючи з 0. При описі елементів масиву можна привласнити значення:

```
int x [10] = {20, 2, 5, 89, 7, 5, 3, 6, 1, 4};
```

2.1.1 Дії над елементами масивів.

Доступ до елементів масиву здійснюється шляхом вказівки імені змінної масиву, за яким в квадратних дужках міститься значення індексу (порядкового номера) елемента. Приклади завдання індексу: M [5] - безпосередньо числом; M [k] - побічно через змінну k; M [k1 + 5] - побічно через вираз k1 + 5; M [abs (iM)] - побічно через значення функції.

Наприклад, за допомогою оператора

```
mas [2] = 34;
```

елементу масиву **mas** з індексом 2 присвоюється значення 34. Оператор

```
cout << mas [5] << endl;
```

виведе на екран значення елемента масиву mas з номером 5.

Всім елементам одного масиву можна привласнити значення елементів іншого масиву за допомогою одного оператора присвоювання. Так, якщо задані такі масиви:

```
int x [10], y [10];
```

то неприпустимий наступний оператор присвоювання:

```
x = y; //
```

У мові C над масивами не визначені операції відносини. Порівнювати масиви можна тільки поелементно. До окремих елементів масиву можна застосовувати стандартні процедури і функції, передбачені в мові. Перелік допустимих стандартних підпрограм залежить від типу елементів масиву.

Використовуючи в якості індексу змінну циклу for можна звернутися до кожного елемента масиву по черзі. Так за допомогою фрагмента програми

```
for (i = 1; i <= 25; i++) mas [i] = 0;
```

всіх елементів масиву присвоюється нульове значення.

Використання масивів замість одиночних змінних дає перевагу при обробці багатьох однотипних даних. Нехай, наприклад, метеорологічна станція виробляє вимір температури повітря щодня в один і той же час. Потрібно знайти середньомісячну температуру. Використання масивів дозволяє об'єднати дані вимірювань в один масив, а також компактна запрограмувати їх обробку.

Приклад.

Дана послідовність температур $t_0 \dots t_{30}$. Організувати масив для зберігання цієї послідовності. Визначити середньомісячну температуру.

```
1. #include <iostream.h>
2. int main(void)
3. {
4. float t[31];    //описание массива
```

```

5. int i;    //параметр цикла for
6. float s; //сумма элементов
7. for (i=0;i<=30;i++) //заполнение массива
8. {
9. cout << "введите элемент с номером" << i;
10.    cin >> t[i];
11. };
12.    s=0; //обнуление суммы
13.    for (i=0;i<=30;i++) s=s+t[i];
    //вычисление суммы
14.    cout << "среднемесячная температура" <<
    s/31; }
15.    // конец программы.
16.

```

2.2 Зв'язок покажчиків і масивів

Існує тісний зв'язок між масивами і вказівниками. Вона полягає в тому, що в оголошеному масиві його ім'я є вказівником на масив, а точніше, на перший елемент масиву. Таким чином, якщо був оголошений масив `int t [31];` то `t` є вказівником на масив, а оператори `p1 = t;` і `p1 = & t [0];` приведуть до одного і того ж результату. Для того щоб отримати значення 8-го елемента масиву `t` можна написати `t [7]` або `* (t + 7)`. Результат буде один і той же. Перевага другого варіанта полягає в тому, що арифметичні операції над вказівниками виконуються швидше, ніж дії з елементами масиву. Як приклад перепишемо попередню програму обчислення середньомісячної температури з використанням вказівників.

```

1. #include <iostream.h>
2. int main(void)
3. {
4. float t[31];    //описание массива
5. int i;    //параметр цикла for
6. float s; //сумма элементов
7. for (i=0;i<=30;i++) //заполнение массива

```

```

8. {
9. cout << "введите элемент с номером" << i;
10.    cin >> *(t+i);
11.    };
12.    s=0; //обнуление суммы
13.    for (i=0;i<=30;i++) s=s*(t+i);
    //вычисление суммы
14.    cout << "среднемесячная температура" <<
    s/31; }
15.    // конец программы.

```

2.3 Заповнення масивів

Робота з масивами вимагає їх попереднього заповнення, тобто присвоювання початкового значення кожного елементу масиву. Для заповнення масиву можна, наприклад, використовувати введення з клавіатури:

```

1. cout << "Введите значения для элементов
    массива:\n";
2. for (i=0;i<=99;i++)
3. {
4. cout << "x1[" << i << "]=";
5. cin >> x1[i];
6. }; //.

```

Однак при налагодженні програми доводиться вводити з клавіатури багато елементів масиву. Щоб позбутися від цієї виснажливої роботи доцільно заповнювати масиви випадковими числами. Для цього кожному елементу масиву присвоюється значення за допомогою датчика псевдовипадкових чисел `rand()`.

Функція `rand()` визначена в бібліотеці `stdlib` і вміє генерувати псевдовипадкове ціле число між 0 і константою `RAND_MAX`, певної в тій же бібліотеці. Перед використанням датчик випадкових чисел необхідно ініціалізувати початковим значенням. Для цього призначена функція `srand(unsigned long int)`, яка отримує цілий позитивний аргумент і задає початкове число для функції `rand()`. Аргумент

функції `srand ()` можна вводити з клавіатури. Тоді при введенні однакових значень можна отримати однакові послідовності випадкових чисел, що корисно для налагодження програми. При введенні різних значень аргументу функції `srand ()` генеруються послідовності випадкових чисел будуть різними. Якщо вводити з клавіатури початкове число небажано, то можна зчитувати його з системного таймера, наприклад, так `srand (time (NULL))`. Функція `time ()` належить бібліотеці `time`. Таким чином, для заповнення масиву цілими числами від 0 до N можна використовувати фрагмент програми

```
1. #include<stdlib.h>
2. #include<time.h>
3. ...
4. srand(time(NULL)); //Инициализируем датчик
5. for (i=0;i<=99;i++) //Для каждого элемента
6. x1[i]=rand()%(N+1); //Присваиваем случайное
```

Тут використана операція взяття залишку від ділення для приведення діапазону значень випадкової величини до відрізка `[0; N]`. Якщо необхідно заповнити масив речовими псевдовипадковими числами з відрізка `[a; b]`, можна писати так:

```
x1[i]=a+(b-a)*1.0*rand()/RAND_MAX; //.
```

Тут результат ділення `rand()/RAND_MAX` за змістом є речовим випадковим числом з відрізка `[0; 1]`. Однак компілятор C (C++) при розподілі цілого на ціле не виконує приведення типу результату до матеріального числа. Тому в вираз введено множення на речову одиницю 1: 0. Оскільки в правій частині виразу зустрівся хоч один дійсний множник, результат буде приведений до матеріального типу. перетворення

```
y =a+(b-a) * x дозволяє трансформувати діапазон значень x2[0; 1] в діапазон значень y2[a;b] .
```

2.4 Пошук елементів, що задовольняють заданій умові.

Найчастіше доводиться виконувати пошук в нерегульованих масиві елементів, які задовольняють заданій умові, наприклад, умовою позитивності. Наступна програма заповнює масив цілими

випадковими числами від -100 до 100, визначає кількість позитивних елементів і видає повідомлення на екран.

```

1. #include <iostream.h>
2. #include <stdlib.h>
3. #include <time.h>
4. int main(void)
5. {
6.     int m[30];          //Описание массива
7.     int i;              //параметр цикла for
8.     int n;              //счётчик положительных элементов
9.     srand(time(NULL));  //Инициализация датчика
10.    for (i=0; i<=29;i++) //заполнение
        массива
11.        m[i]=rand()%201-100;
12.    n=0; //обнуление счётчика элементов
13.    for (i=0; i<=29;i++)
14.        if (m[i]>0)      //Условие
            положительности
15.            n++; //Наращиваем на 1 счетчик
16.    cout << "в массиве " << n << "
        положительных эл-ов"; return 0;
17.    } // конец программы.
18.

```

Для підрахунку кількості елементів, що задовольняють умові позитивності введена змінна n з нульовим початковим значенням. Далі циклічно виконується почергове порівняння кожного елемента масиву з нулем. У разі позитивності чергового елемента змінна n нарощується на одиницю.

2.5 Визначення максимального елемента і його положення в масиві

Розглянемо приклад програми, яка заповнює матеріальний масив випадковими числами і визначає значення і індекс (номер) максимального елемента цього масиву

```

1. #include <iostream.h>

```

```

2. #include <stdlib.h>
3. #include <time.h>
4. int main(void)
5. {
6.     float m[30];    //Описание массива
7.     int i;          //параметр цикла for
8.     float max;       //значение максимального
        элемента
9.     int t;          // индекс (номер) макс. элемента
10.    srand(time(NULL)); //Инициализация
        датчика
11.    for (i=0; i<=29;i++) //заполнение
        массива
12.        m[i]=10.0*rand()/RAND_MAX;
13.        // допустим, что 1-й элемент максимален
14.        max=m[0];
15.        t=0;
16.        for (i=1; i<=29;i++) //все остальные
        элементы
17.            if (m[i]>max)    //сравниваем с
        максимальным
18.
19.
20.            {
21.                max=m[i];
22.
23.                t=i;
24.            };
25.    cout << "максимальный элемент = " <<
        max;
26.    cout << "номер максимального элемента"
        << t;
27.    return 0;
    } //конец программы.

```

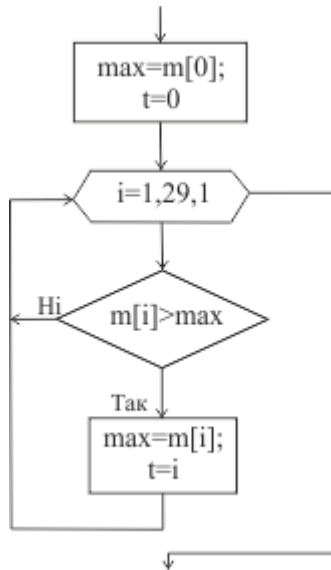


Рисунок 2.1 – Блок-схема алгоритму пошуку максимального елемента масиву

Алгоритм пошуку максимального елемента досить простий. Для зберігання значення максимального елемента передбачимо змінну max того ж типу, що й елементи масиву, а для зберігання індексу максимального елемента - змінну цілого типу t . Припустимо спочатку, що перший елемент масиву (з номером 0) є максимальним: $max = m[0]$; $t = 0$;. Далі по черзі порівнюємо кожен елемент масиву з поточним значенням змінної max . Якщо черговий елемент перевищив максимальне значення, то присвоюємо змінній max значення цього елемента, а змінній t значення його номера. На рис. 2.1 показаний фрагмент блок-схеми описаного алгоритму. Якщо масив містить кілька елементів з однаковим максимальним значенням, то наведена програма знайде тільки перший з них. Аналогічно виконується пошук мінімального елемента масиву.

2.6 Упорядкування (сортування) масивів

У більшості випадків масиви застосовуються для зберігання великої кількості однотипних даних, створення і організації баз і банків даних тощо У багатьох випадках номер (індекс) елемента в

масиві не грає великої ролі. Важливо саме значення елемента, тобто сама інформація. Пошук потрібної інформації в неврегульованих масиві або банку даних можливий тільки способом послідовного перебору всіх елементів. Існують алгоритми швидкого пошуку інформації, але всі вони працюють тільки з впорядкованими масивами даних. Тому завдання сортування елементів масивів є дуже актуальною.

Нехай дано числовий масив x з n попарно різних елементів. Потрібно переставити елементи масиву так, щоб вони були впорядковані по зростанню

$$x[0] < x[1] < \dots < x[n-1]:$$

Існує багато алгоритмів вирішення цього завдання.

Алгоритм сортування перебором.

Очевидно, що перше місце в масиві повинен зайняти мінімальний елемент масиву, друге - найменший з усіх інших, третє - найменший з решти. Отже, для сортування необхідно виконати наступну послідовність дій:

- 1) визначити мінімальний елемент масиву;
- 2) поміняти його місцями з першим елементом;
- 3) визначити мінімальний елемент серед решти;
- 4) поміняти його місцями з другим елементом і т.д.

Ця послідовність дій повинна виконуватися до тих пір, поки не буде визначено останній мінімальний елемент.

Всю операцію по упорядкуванню масиву можна розбити на більш прості завдання. Перша пошук мінімального елемента серед елементів з номерами $i..n$. Величина i повинна бути рівна спочатку 1, потім 2, 3 і т.д до $n-1$. Друга заміна місцями мінімального елемента з номером t і елемента номером i . Поміняти місцями дві змінні x і y можна двома способами:

- 1) з використанням допоміжної змінної $v = x; x = y; y = v$;
- 2) без використання допоміжної змінної $x = x + y; y = x - y; x = x - y$; але тільки для числових даних.

На рис. 2.2 зображена блок-схема алгоритму сортування масиву методом перебору.

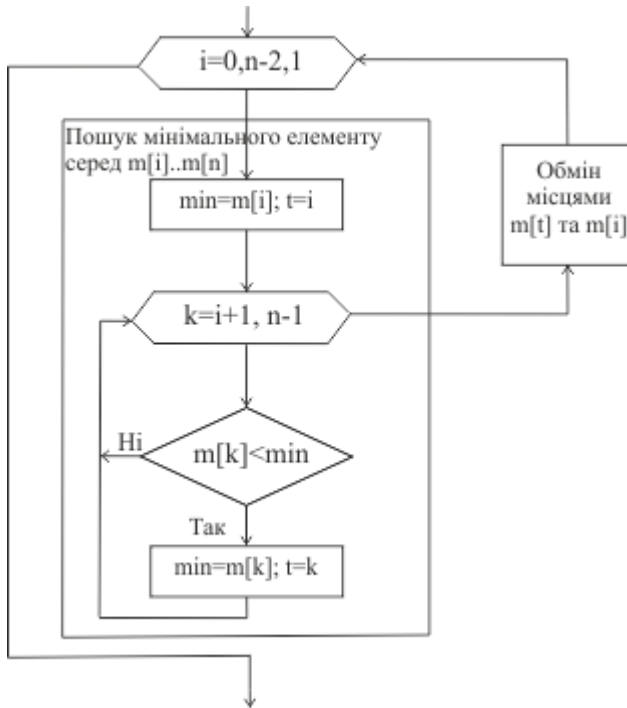


Рисунок 2.2 – Блок-схема алгоритму сортування масиву методом перебору

Алгоритм сортування методом бульбашки.

Даний спосіб сортування заснований на послідовному порівнянні сусідніх елементів масиву. Якщо два сусідні елементи розташовані не в потрібній послідовності, то міняємо їх місцями. Так повторюємо до тих пір, поки в черговому проході не зробимо жодного обміну, тобто масив буде впорядкованим. На рис. 18 показана блок-схема алгоритму сортування за зростанням методом бульбашки (m - масив для сортування з початковим індексом 0, n - розмір масиву).

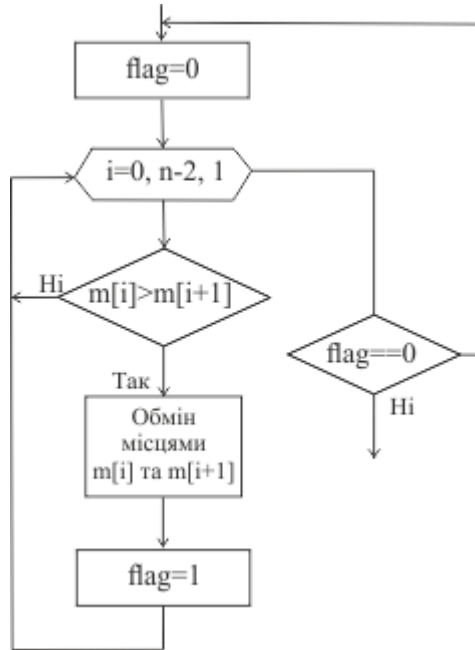


Рисунок 2.3 – Блок-схема алгоритму сортування масиву методом бульбашки

Як приклад відсортуємо по зростанню масив

5 2 7 4 3 1 8 6.

Перший прохід по масиву призводить до обміну місцями елементів 5 і 2, 7 і 4, 7 і 3, 7 і 1, 8 і 6:

2 5 7 4 3 1 8 6,

2 5 4 7 3 1 8 6,

2 5 4 3 7 1 8 6,

2 5 4 3 1 7 8 6,

2 5 4 3 1 7 6 8.

В результаті останнє місце зайняв максимальний елемент 8. Протягом другого проходу міняються місцями 5 і 4, 5 і 3, 5 і 1, 7 і 6:

2 4 5 3 1 7 6 8,

2 4 3 5 1 7 6 8,

2 4 3 1 5 7 6 8,

2 4 3 1 5 6 7 8.

Третій прохід призводить до обміну 4 і 3, 4 і 1:

2 3 4 1 5 6 7 8,

2 3 1 4 5 6 7 8

четвертий прохід 3 і 1:

2 3 4 1 5 6 7 8,

2 1 3 4 5 6 7 8,

і, нарешті, п'ятий прохід 2 і 1:

1 2 3 4 5 6 7 8.

В результаті для сортування цього масиву треба було 5 проходів. У загальному випадку кількість необхідних проходів залежить від ступеня "сортування" вихідного масиву, але не перевищує $n - 1$.

Алгоритми швидкого сортування.

Метод "швидкої" сортування, запропонований К. Хоаром (С. Hoare), заснований на поділі масиву на два непустих непересічних підмножини елементів. Для цього вибирається елемент масиву (наприклад, розташований посередині), запам'ятовується його значення.

Далі елементи переставляються так, щоб в одній частині масиву (наприклад спочатку) виявилися всі елементи, значення яких не перевищують значення обраного елемента, в іншій частині - всі елементи, не менші його. Тоді вибраний елемент виявиться десь посередині масиву і буде служити межею поділу частин. При цьому кожна з двох частин масиву не є відсортованою. Аналогічно слід вчинити з кожної з отриманих частин і так далі. На певному етапі масив виявиться повністю відсортованим. Як правило використання "швидкої" сортування дає кращі результати за швидкістю в порівнянні з усіма іншими методами.

Оскільки алгоритм поділу на частини однаковий як для всього масиву, так і для його частин, доцільно зробити його залежним від аргументів початкового і кінцевого індексу сортованої частини масиву. Тоді один і той же алгоритм можна застосувати як для упорядкування всього масиву, так і для його частин. Отже "швидке" сортування найзручніше формулювати в рекурсивному вигляді. Дійсно, алгоритм сортування всього масиву після поділу його на частини повинен викликати сортування частин із зазначенням початкового та кінцевого індексів кожної частини.

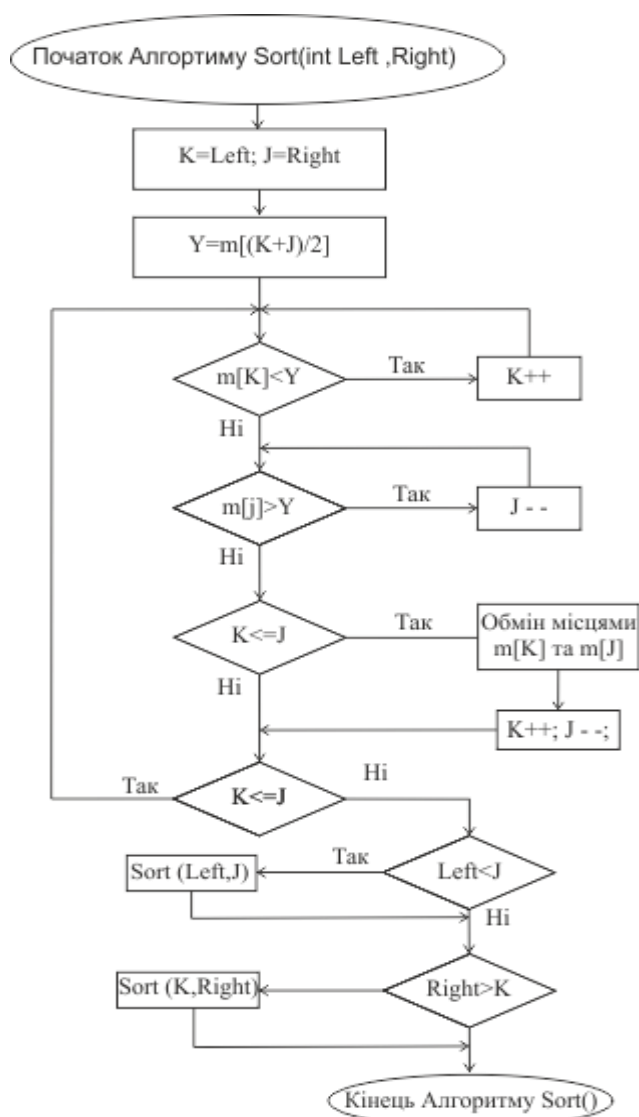


Рисунок 2.4 – Блок-схема алгоритму швидкого сортування масиву

Якщо відомо, що масив не містить повторюваних елементів, то елемент, що розділяє масив на частини можна вибирати довільно. Блок-схема алгоритму швидкого сортування масиву з попарно

різними елементами показана на рис. 2.4. Вихідними даними для нього є величини Left номер елемента, який вважається лівої кордоном сортованої частини масиву і Right номер елемента, який вважається її правої кордоном.

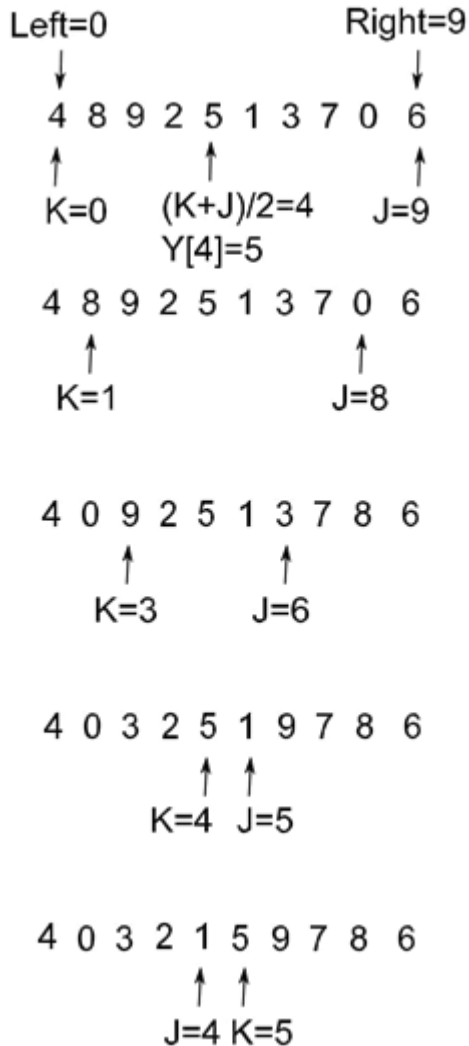


Рисунок 2.5 – Приклад швидкого сортування масиву, крок 1

Спочатку вводяться три локальні змінні: K з початковим значенням лівої межі, J з початковим значенням правої межі і Y значення елемента, розташованого посередині масиву. Оскільки K і J є змінними цілого типу, то значення вираз $(K + J) = 2$ за правилами мови C теж приводиться до цілого типу. Так, наприклад, для масиву, зображеного на рис. 20, $Y = 5$.

Припустимо, що необхідно упорядкувати елементи масиву по зростанню. Будемо збільшувати на 1 змінну K до тих пір, поки не знайдеться елемент $m[K] \leq Y$. Змінну J будемо зменшувати на 1 до тих пір, поки не знайдеться елемент $m[J] \geq Y$. Для масиву, показаного на рис. 20, це відповідно елементи $m[1] = 8$ і $m[8] = 0$. Якщо KJ , то міняємо знайдені елементи масиву місцями, збільшуємо на 1 змінну K і зменшуємо на 1 змінну J . Якщо KJ , продовжуємо збільшувати на 1 змінну K до тих пір, поки не знайдеться елемент $m[K] \leq Y$, а змінну J зменшувати на 1 до тих пір, поки не знайдеться елемент $m[J] \geq Y$. Для масиву, показаного на рис. 20, наступними знайденими елементами є $m[3] = 9$ і $m[6] = 3$. Міняємо їх місцями і аналогічно знаходимо такі елементи масиву, що підлягають обміну: $m[4] = 5$ і $m[5] = 1$. Так відбувається до тих пір, поки не виконається умова $J \geq K$. Стосовно до масиву рис. 20 $J = 4$, $K = 5$. Ці значення змінних J і K визначають межі поділу масиву на 2 частини $[Left, J]$ і $[K, Right]$. Далі алгоритм сортування викликається для кожної з цих частин окремо.

На рис. 2.6. показаний другий крок процес сортування лівої і правої частин масиву. В результаті кожна з цих частин буде поділена ще на дві частини (елементи з номерами 0,1,2; 3,4; 5,6; 7,8,9), для кожної з яких буде викликаний алгоритм сортування.

Якщо вихідний масив може містити однакові елементи, то елемент, що розділяє масив на частини вже не можна вибирати довільно. Наприклад, масив (або його частина), що складається з елементів 8 7 8 9 8, призведе до зациклення алгоритму сортування, так як крайні елементи і середній розділяє елемент рівні один одному $m[Left] = m[Right] = Y$. У цьому випадку замість простого вибору $Y = m[(K + J) / 2]$ необхідно застосовувати більш витончені способи вибору розділяє елемента. Доцільно вибирати його близьким до середнього арифметичного всіх елементів масиву і нерівним першому і останньому елементам.

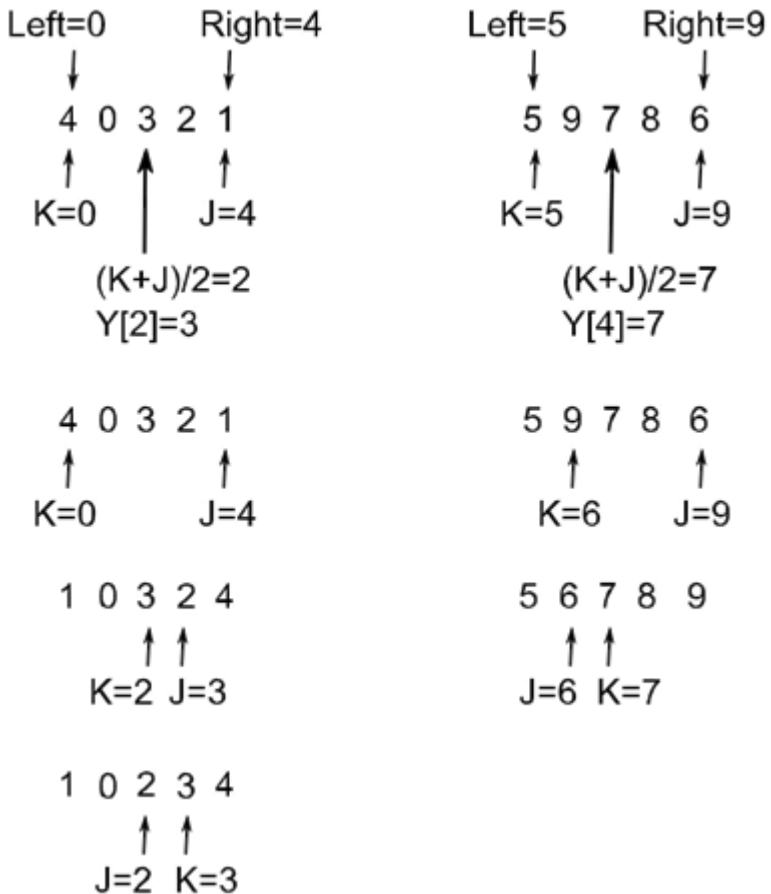


Рисунок 2.6 – Приклад швидкого сортування масиву, крок 2

2.7 Видалення елементів з масиву і вставка елементів в масив

До сих пір розглядалися масиви, розмір яких відомий заздалегідь до компіляції програми. С допомогою динамічного виділення пам'яті досить просто створити масив, розмір якого

невідомий заздалегідь. Нехай, наприклад, розмір масиву n вводиться з клавіатури, потім виділяється пам'ять для всього масиву:

```
int n;
cin >> n // ввод размера массива
...
int *massiv = new int[n]; // выделение памяти.
```

Тут в одному рядку оголошується покажчик на ціле `massiv` і виділяється пам'ять для n елементів масиву. Здавалося б цей масив не має імені, однак покажчик на його перший елемент `massiv` і є ім'я масиву. Отже доступ до елементів можливий як через розіменування покажчика `* (massiv + k)`, так і через ім'я `massiv [k]`. На жаль, стандартна мова С не має коштів для створення масивів змінної довжини. В ході виконання програми довжина масиву змінюватися не може. Тому при необхідності роботи з масивами змінної довжини описують масив свідомо більшого розміру. Разом з тим описують цілу змінну, сенс якої фактична довжина масиву, тобто розмір використовуваної його частини. Хоча цей шлях і нераціональний, ми будемо використовувати його для роботи з масивами змінної довжини, щоб вивчити алгоритми видалення і вставки елементів. Щоб видалити елемент з номером g з масиву `mas`, необхідно кожен елемент, починаючи з $g + 1$ і до останнього, перемістити на місце попереднього елемента і зменшити на 1 розмір масиву. Описаний алгоритм реалізується наступним фрагментом програми:

```
float mas[100]; //Описание массива
int i; //параметр цикла for
int n; //Фактический размер массива
.....
for (i=g; i<=n-1; i++) mas[i]=mas[i+1];
n--; //Уменьшение n на 1.
```

Для вставки елемента в масив на місце з номером g необхідно попередньо звільнити для нього місце. Для цього починаючи з кінця масиву і до елемента з номером g кожен елемент переміщується на наступне за ним місце, і розмір масиву збільшується на 1. Потім присвоюється нове значення елементу з номером g :

```
1. for (i=n-1; i>=g; i--) mas[i+1]=mas[i];
```

2. `mas[g]=...`
3. `n++;` //Увеличение n на 1

2.8 Багатовимірні масиви

Елементи багатовимірних масивів нумеруються не одним, а кількома (як мінімум двома) індексами. Наприклад, масив `float x2 [10] [20]` являє собою двовимірний масив, який можна інтерпретувати як матрицю з 10 рядків і 20 стовпців. Елементи цього масиву можуть приймати дійсні значення.

Масив `float x3 [10] [20] [5]` є тривимірний масив з 1000 елементів. Найчастіше в програмах не використовують масивів розмірністю більше 2. Оскільки двовимірні масиви за змістом є матрицями, їм слід приділити більше уваги. Двовимірний масив структура даних, що зберігає прямокутну матрицю. У матриці кожен елемент визначається номером рядка і номером стовпця, на перетині яких він розташований. У C ++ двовимірний масив представляється масивом, елементами якого є одновимірні масиви. Доступ до кожного окремого елементу здійснюється зверненням до імені масиву із зазначенням індексів (перший індекс номер рядку, другий індекс номер стовпчика). Всі дії над елементами двовимірного масиву ідентичні діям над елементами одновимірного масиву. Тільки для ініціалізації двовимірного масиву використовується вкладений цикл `for`, наприклад

1. `for (i=0; i<=9; i++)`
2. `for (j=0; j<=19; j++)`
3. `A[i][j] = 0;       //.`

Для ілюстрації принципів роботи з двовимірними масивами розберемо кілька прикладів.

Приклад 1.

Сформувати таблицю Піфагора (таблиця множення) і вивести її на екран.

1. `#include <iostream.h>`
2. `#include <math.h>`
3. `int main(void)`
4. `{`
5. `int p[9][9];`

```

6. int i, j;
7. for (i=0; i<=8; i++) //Цикл по строкам
8. for (j=0; j<=8; j++) //Цикл по столбцам
9. p[i][j]= (i+1)*(j+1);
10.     for (i=0; i<=8; i++)
11.     {
12.         for (j=0; j<=8; j++) cout << p[i][j] <<
            "\t"; cout << endl;
13.         //Перевод курсора на сл. строку }
14.     }

```

Приклад 2.

Заданий двомірний масив `b [10] [10]` цілих чисел, заповнений випадковими числами з відрізка `[-10,10]`. Знайти та вивести на екран ті елементи масиву, які більше заданого числа `k`.

```

1. #include <iostream.h>
2. #include <math.h>
3. #include <stdlib.h>
4. #include <time.h>
5. int main(void)
6. {
7.     int b[10][10];
8.     int i, j, k;
9.     srand(time(NULL));
10.    for (i=0; i<=9; i++)
11.    {
12.        for (j=0; j<=9; j++)
13.        {
14.            [i][j]=rand()%21-10;
15.            cout << b[i][j] << " ";
16.        };
17.        cout << endl;
18.    };
19.    cout << "Введіть число k";
20.    cin >> k;
21.    for (i=0; i<=9; i++)
22.        for (j=0; j<=9; j++)

```

```

23.             if (b[i][j]>k) cout << b[i][j]
                << "\t"; cout << endl;
24.             } //.
```

Приклад 3.

Задати та роздрукувати речовинний масив 10 на 10, що складається з цілих випадкових чисел в інтервалі [1,100]. Знайти суму елементів, що лежать вище головної діагоналі.

Головною є діагональ, проведена з лівого верхнього кута масиву в правий нижній. При цьому виходить, що елементи, що лежать на головній діагоналі матимуть однакові індекси, а для елементів вище головної діагоналі номер стовпчика буде завжди перевищувати номер рядка.

```

1. #include <iostream.h>
2. #include <math.h>
3. #include <stdlib.h>
4. #include <time.h>
5. int main(void)
6. {
7.     float A[10][10];
8.     int i, k;
9.     float S;
10.    S=0;
11.    srand(time(NULL));
12.    for (i=0; i<=9; i++)
13.    {
14.        for (k=0; k<=9; k++)
15.        {
16.
17.            A[i][k]=(99.0*rand()/RAND_MAX)+1.0;
18.            cout << A[i][k] << " ";
19.            if (k>i) S=S+A[i][k];
20.        };
21.        cout << endl;
22.    };
```

```

21.         cout << "Сумма элементов выше главной
           диагонали равна" << S;
22.     }    // .

```

2.9 Масиви вказівників

Показчики як і змінні будь-якого іншого типу, можуть об'єднуватися в масиви. Оголошення масиву показчиків з 10 елементів має вигляд `int * x [10];`. Кожному з елементів масиву можна привласнити адресу. Наприклад, п'ятого елементу цього масиву дамо адресу цілої і раніше оголошеної змінної `y`: `x [4] = & y;`. Якщо потім необхідно знайти значення змінної `y`, це можна зробити, використовуючи конструкцію `* x [4]`.

За допомогою масиву показчиків нескладно запрограмувати динамічне створення двовимірних масивів. Оголошуємо масив 10 вказівників `p: int *p[10];`

з кожним з них можна зв'язати динамічно створений одновимірний масив і інтерпретувати його як рядок матриці:

```

1. for (i=0; i<=8; i++) //Цикл по строкам
2. {p[i]=new int[20];}   // Создание строки

```

В результаті отримали наступну хитромудру конструкцію. Ідентифікатор `p` є ім'ям масиву показчиків, які містять адреси перших елементів інших масивів рядків матриці. Вираз `p [k]` або рівносильне йому `* (p + k)` є адреса першого елемента рядки з номером `k`. Вираз `* (p [k] + m)` або `* (* (p + k) + m)` елемент з номером `m` рядки з номером `k`, тобто елемент двовимірного масиву, розташований в рядку `k` і стовпці `m`. Зауважимо також, що динамічно створені "рядки матриці" не обов'язково повинні мати однаковий розмір, а саме поняття "матриця" в даному контексті доречно замінити виразом "масив масивів".

2.10 Практичне завдання

При виконанні практичного завдання дотримуватися такої послідовності дій.

- 1) Обов'язково вивчіть опис, пропонуване вище.
- 2) Розберіть приклади, що наводяться в описі.

- 3) Осмисліть завдання, запропоновану в вашому варіанті.
- 4) Складіть словесний алгоритм вирішення задачі.
- 5) Уявіть алгоритм рішення у вигляді блок-схеми.
- 6) Переведіть алгоритм на мову програмування.

2.10.1 Варіанти завдань

Варіант 1.

1) В одновимірному масиві переставте місцями елементи, що знаходяться на парних і непарних місцях. Початковий і перетворений масиви виведіть на екран.

2) В двовимірному масиві підрахуйте і виведіть на екран максимальні і мінімальні елементи кожного рядка. Розрахуйте значення логарифма елемента $[1,1]$ і додавайте до нього послідовно всі залишилися елементи 1-го рядка, потім всі елементи 2-го рядка, починаючи з 1-го елемента, і до тих пір, поки початкове значення (логарифма) не збільшиться в 5 разів. Виведіть на екран індекс елемента, на якому процес закінчиться.

Варіант 2.

1) Перетворіть одновимірний масив таким чином, щоб сума елементів в його першій половині була б якомога ближче до суми елементів його другої половини. Вихідний і перетворений масиви виведіть на екран.

2) В заповненому випадково двовимірному масиві $a[n][m]$ замініть його елементи числами, що обчислюються за формулою $a[i][j] = a[i][j] * \arccos(i / (i + j))$. Для кожного стовпця знайдіть кількість позитивних і негативних елементів в ньому. Елементи початкового і одержаного масивів вивести на екран.

Варіант 3.

1) Елементи випадково заповненого одновимірного масиву $d[n]$ перетворіть за формулою $d[i] = \exp(i / d[i])$. У новому масиві знайдіть і виведіть на екран суму елементів, порядковий номер яких ділиться на 3.

2) Елементи двовимірного масиву $a[m][n]$ обчислюються за формулою $a[i][j] = ij$, якщо $i \neq j$ і $a[i][j] = ji$ в іншому випадку. У цьому масиві знайдіть і виведіть на екран квадратну матрицю $m[3][3]$, сума діагональних елементів в якій мінімальна.

Варіант 4.

1) Зіставте одновимірному масиву $a[n]$ інший одновимірний масив $f[n]$, елементи якого обчислюються за формулою $f[i] = \ln(a[i]) = \arctg(ei = 10)$. Елементи цього масиву переставте таким чином, щоб меншому номеру відповідав би менший елемент. Елементи всіх масивів виведіть на екран в один рядок.

2) В двовимірному масиві знайдіть і виведіть на екран 5 найбільших елементів із зазначенням їх індексів.

Варіант 5.

1) В одновимірному масиві переставте елементи таким чином, щоб на 5-му місці стояло б найбільше, а на 6-му найменше число.

2) В двовимірному масиві $b[n][m]$ знайдіть 4 послідовні елемента, сума яких максимальна. Обхід для пошуку потрібно виконувати по ходу годинникової стрілки, починаючи з елемента $b[0,0]$.

Варіант 6.

1) З'ясуйте, чи є серед елементів двовимірного масиву прості числа. Повідомте їх індекси, підрахуйте загальну кількість і середнє значення.

2) В одновимірному масиві знайдіть середнє значення всіх елементів і побудуйте новий одновимірний масив з різниць відповідних елементів і отриманого середнього. Результати виведіть на екран.

Варіант 7.

1) Замініть в одновимірному масиві кожен з елементів сумою його подільників.

2) В двовимірному масиві знайдіть квадрат розміром 3×3 , в якому сума діагональних елементів максимальна. Виведіть його на екран.

Варіант 8.

1) Кожен з елементів одновимірного масиву замініть сумою цифр цього елемента. Знайдіть максимальний і мінімальний елементи. Вихідний і перетворений масиви виведіть на екран.

2) Перетворіть вихідний двовимірний масив $a[m][n]$ в одновимірний $f[m * n]$ і, вважаючи кожен з елементів довжиною

сторони трикутника, знайдіть трійку таких елементів, які утворюють трикутник з найбільшою площею.

Варіант 9.

1) У двовимірному масиві замініть кожен елемент часткою від ділення самого елемента на суму його дільників.

2) Знайдіть всі цифри в елементах одновимірного масиву і підрахуйте, скільки разів зустрічається кожна.

Варіант 10.

1) Елементи одновимірного масиву перетворіть в новий одновимірний масив, в якому кожен елемент дорівнює найбільшому дільнику вихідного, а елементи прості числа залишені без змін. Обидва масиви виведіть на екран.

2) Для кожного рядка двовимірного масиву підрахуйте і виведіть на екран кількість елементів, кратних 3 і кратних 7.

Варіант 11.

1) У двовимірному масиві переставте рядки таким чином, щоб середнє значення елементів кожного рядка зростала в міру зростання її номера.

2) З'ясуйте, чи містяться в одновимірному масиві вчинені числа. Перетворіть цей масив таким чином щоб всі елементи, які не є досконалими числами, були замінені сумами їх подільників. Вихідний і перетворений масиви виведіть на екран.

Варіант 12.

1) Створіть одновимірний цілочисельний масив. Упорядкуйте масив по зростанню елементів методом бульбашки.

2) Знайдіть добуток двох цілочисельних матриць 3×3 . В отриманій матриці перестановкою рядків і стовпців добийтеся, щоб мінімальний елемент мав координати [1] [1].

Варіант 13.

1) Сформуйте речову матрицю 5×5 . Обчисліть добуток негативних елементів, що знаходяться над і під головною діагоналлю матриці.

2) Упорядкуйте одновимірний речовинний масив з 20 елементів по спадаючій методом перебору.

Варіант 14.

1) Сформуйте речову матрицю 5×5 . Упорядкуйте по зростанню елементи кожного рядка матриці.

2) Розрахуйте суму квадратів непарних елементів цілочисельного масиву з 20 елементів.

Варіант 15.

1) Введіть цілочисельний двовимірний масив з N рядків і N стовпців. Знайдіть максимум серед сум елементів діагоналей, паралельних побічної діагоналі.

2) Задано одновимірний масив з N дійсних чисел. Визначте утворюють чи елементи масиву зростаючу послідовність. Якщо послідовність не збільшується, то впорядкуйте її за зростанням будь-яким способом.

Варіант 16.

1) Задано одновимірний масив. Визначте скільки разів змінюється знак в даному масиві, визначте номера позицій, в яких змінюється знак.

2) Упорядкуйте матрицю по рядках по зростанню.

Варіант 17.

1) задані два одновимірних масиву з N і M елементами, впорядкованими по не спаданню, об'єднайте елементи цих масивів в один масив так, щоб елементи знову виявилися впорядкованими по не спаданню.

2) Здійсніть поворот матриці проти годинникової стрілки на 90 градусів.

Варіант 18.

1) Дан одновимірний масив A , що складається з N елементів. З'ясуйте, скільки значень елементів в масиві A зустрічається більше одного разу?

2) Визначте, чи є задана матриця ортонормованою, тобто скалярний добуток кожної пари різних рядків (або стовпчиків) дорівнює нулю.

Варіант 19.

1) Дан одновимірний масив A , що складається з N елементів. Підрахуйте максимальну кількість поспіль нулів.

2) Заповніть квадратну матрицю В [10] [10] одиницями і нулями в шаховому порядку.

Варіант 20.

1) Дан одновимірний масив А, що складається з N елементів. Виключіть з масиву перший позитивний елемент, наступний за максимальним.

2) Для матриці 3 [m] [n], знайдіть суму елементів, великих 3,2. Переставте стовпці по зростанню кількості негативних елементів у стовпцях.

Варіант 21.

1) Дан одновимірний масив А, що складається з N елементів. Підрахуйте максимальну кількість позитивних елементів, укладених між нулями.

2) Сформууйте матрицю з трикутником Паскаля (бі-номінальні коефіцієнти)

1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	2	1	0	0	0	0	0
1	3	3	1	0	0	0	0
1	4	6	4	1	0	0	0
1	5	10	10	5	1	0	0
1	6	15	20	15	6	1	0

Варіант 22.

1) Дан цілочисельний масив М [15]. Визначте число сусідства з двох чисел різного знаку.

2) Для масиву S [m] [n] визначте рядок з найбільшою кількістю негативних елементів. Переставте стовпці по зростанню суми позитивних елементів стовпців.

Варіант 23.

1) Визначте в даному масиві Р [m] кількість пар сусідніх чисел, які є протилежними.

2) Дана матриця А [m] [n]. Визначте середнє арифметичне позитивних елементів матриці. Переставте стовпці по зростанню значення останнього елемента стовпців.

Варіант 24.

- 1) Заданий масив A [10]. Визначити, скільки міститься в ньому різних чисел.
- 2) Дана матриця A [m] [m]. Знайдіть мінімальний з елементів, розташованих під головною діагоналлю. Переставте стовпці по зростанню значення другого елементу стовпчика.

Варіант 25.

- 1) Дан масив B [n], який містить велику кількість нульових елементів. Замініть всі групи поспіль де зустрічаються нулів на один нуль.
- 2) Дана матриця A [m] [m]. Знайдіть максимальний з елементів, розташованих над головною діагоналлю. Переставте рядки по зростанню значення другого елементу рядка.

Варіант 26.

- 1) Дан масив X [K], який містить велику кількість нульових елементів. Замініть групи елементів, що складаються з непарної кількості нулів, на один нульовий елемент, а з парного на два.
- 2) Знайдіть всі цифри в елементах двовимірного масиву і підрахуйте, скільки разів зустрічається кожна.

Варіант 27.

- 1) Дан масив Y [n], який містить велику кількість нульових елементів. Замініть всі групи поспіль зустрічаються нулів на елемент, що складається з двох цифр, де перша цифра 0, а друга кількість нулів в групі.
- 2) Упорядкуйте двовимірний масив по стовпчиках по зростанню

3 ЛАБОРАТОРНА РОБОТА № 3 «РОБОТА З ТЕКСТОМ, ФАЙЛАМИ»

Мета роботи: освоєння прийомів роботи з текстовими даними; освоєння прийомів роботи з файлами.

3.1 Робота з рядками

3.1.1 Робота з рядками засобами мови C

Більшість програм містять елементи роботи з текстовими даними. Текстові дані складаються з рядків. Рядок являє собою масив символів. Тому масивів типу `char` символьним масивам слід приділити особливу увагу при вивченні програмування. У багатьох мовах програмування є спеціальний тип даних рядок символів (`string`). У мові C такого типу немає, і робота з рядками реалізована шляхом використання одновимірних масивів типу `char`. У мові C символьний рядок це одновимірний масив типу `char`, що закінчується нульовим байтом. Для нульового байта визначена спеціальна символьна константа `\0`. Це необхідно враховувати при описі відповідного символьного масиву. Так, якщо рядок повинен містити `N` символів, то в описі масиву необхідно вказати `N + 1` елемент. Наприклад, опис `int str [12]`, передбачає, що рядок містить 11 символів, а останній байт зарезервований під нульовий байт.

Хоча в мові C немає спеціального типу рядка, мова допускає рядкові константи. Строкова константа це послідовність літер, взятих в подвійні лапки. Наприклад, "Це строкова константа". В кінець строкової константи не треба ставити символ `\0`. Це зробить сам компілятор, і рядок "CodeBlocks" в пам'яті буде виглядати так:

C	o	d	e	B	l	o	c	k	s	\0
---	---	---	---	---	---	---	---	---	---	----

Є два простих способи ввести рядок з клавіатури. Перший спосіб скористатися функцією `scanf ()` зі специфікатором введення `%s`. Треба пам'ятати, що функція `scanf ()` вводить символи до першого пробілу символу. Другий спосіб скористатися функцією `gets ()`, оголошеної в файлі `stdio.h`. Функція `gets ()` дозволяє вводити рядки, що містять пробіли. Введення закінчується натисканням клавіші `Enter`.

Обидві функції автоматично додають в кінці рядка нульовий байт. Як параметр в цих функціях використовується ім'я масиву.

Висновок рядків виробляється за допомогою функцій `printf ()` і `puts ()`. Обидві функції виводять вміст масиву до першого нульового байта. Функція `puts ()` додає в кінці рядка, що виводиться символ нового рядка. У функції `printf ()` перехід на новий рядок потрібно передбачати в рядку формату самим. Розглянемо приклад:

```

1. # include <stdio.h>
2.      Ввод строки с клавиатуры и вывод ее на
      экран int main(void)
3. {
4. char str[80];
5. printf("Введите строку длиной не более 80
      СИМВОЛОВ");
6. gets(str);
7. printf(" Вы ввели строку %s\n",str);
8. printf("Введите еще одну строку ");
9. scanf("%s",str);
10.
11.      printf(" Вы ввели строку ");
12.      puts(str);
13.      } //.
```

Для роботи з рядками існує спеціальна бібліотека, заголовки якої називається `string.h`. Перерахуємо деякі функції цієї бібліотеки.

Функція `strcpy (s1, s2)` використовується для копіювання вмісту рядка `s2` в рядок `s1`. Рядок `s1` повинна бути досить великою, щоб у неї помістилася рядок `s2`. Якщо місця мало, то компілятор не видасть відповідного повідомлення про помилку, також це не призведе до переривання виконуваної програми, але може привести до псування інших даних, що відіб'ється на результаті.

Функція `strcat (s1, s2)` приєднує рядок `s2` до рядка `s1` і поміщає її в рядок, `s1`, при цьому рядок `s2` не змінюється. Нульовий байт, який завершував рядок `s1`, буде замінений першим символом рядка `s2`. І в функції `strcpy ()`, і в функції `strcat ()` отримана рядок автоматично

завершується нульовим байтом. Розглянемо приклад використання цих функцій:

```

1. # include <stdio.h>
2. # include <string.h>
3. int main(void)
4. {
5.     char s1[20], s2[20]; // описали массивы
6.     strcpy(s1, " Hello, "); // s1="Hello, "
7.     strcpy(s2, " World !"); // s2="World !"
8.     puts(s1); puts(s2); // выводим Hello, World
9.     !
10.    strcat(s1,s2); // s1="Hello, World !"
11.    puts(s1); // выводим Hello, World !
12.    } //.
```

Виклик функції strcmp () має вигляд strcmp (s1, s2). Ця функція порівнює рядки s1 і s2 і повертає нульове значення, якщо рядки рівні, тобто містять одне і те ж число однакових символів. Під порівнянням рядків розуміється порівняння в лексикографічному сенсі. Якщо s1 лексикографічно більше s2, то функція strcmp () повертає позитивне значення, якщо менше негативне.

Функція strlen (s) повертає довжину рядка s, при цьому завершальний нульовий байт не враховується. Наприклад, виклик strlen ("hello") поверне значення 5.

3.1.2 Масиви рядків

Оскільки рядок являє собою масив символів, то масив рядків є двовимірним масивом символів з рядками різної довжини. Для роботи з подібного роду даними зручно використовувати масиви покажчиків. Кожен з вказівників масиву може містити адресу свого масиву символів, тобто своєї рядки. Розглянемо приклад використання масиву рядків, запозичений з [4]:

```

1. #include <stdio.h>
2. #include <string.h>
```

```

3. #include <stdlib.h>
4. #include <conio.h>
5. int main (void)
6. {
7. // описуємо масив рядків
8. char * ext [] = { "exe", "com", "dat", "c",
   "pas", "cpp"};
9. char ch, s1 [80];
10.
11.     for (;;) // нескінченний цикл
12.     {do
13.     {
14.         Виводимо підказки на екран printf (
   "Файли з розширенням: \n"); printf ( "1.
   exe \n");
15.         printf("2. com\n");
16.         printf("3. dat\n");
17.         printf("4. c\n");
18.         printf("5. pas\n");
19.         printf("6. cpp\n");
20.         printf("7. Exit\n");
21.         printf(" Ваш ви́бор: \n");
22.         Ввод ви́бранного символу 1-7
   ch=getche();
23.         printf("\n"); }
24.         пока не буде́т на́жат символ 1-7 while
   ((ch<'1')||(ch>'7'));
25.         анализ введённого символа
26.         if ( ch == '7' ) break;
27.         strcpy(s1,"dir *.");
28.         strcat(s1,ext[ch-49]);
29.         system(s1);
30.     }
31. } //.
```

Дана програма формує командний рядок для виконання команди "dir" (висновок на екран списку файлів поточного каталогу) і потім виконує її за допомогою бібліотечної функції system (). Користувач може вибрати розширення виведених файлів з тих, які надає програма.

Можливі розширення зберігаються у вигляді масиву рядків і виводяться на екран як підказка. Вибір потрібного розширення здійснюється введенням його номера в списку. Далі програма аналізує код введеного символу. Символи цифр від 1 до 7 мають коди 49 ... 55. Отже, введення одиниці відповідає символу з кодом 49 і повинен привести до вибору з масиву розширення "exe" елемент масиву з номером 0. Якщо користувач хоче вивести файли з розширенням srr, він вказує символ 6, код якого дорівнює 54. Різниця 54 - 49 визначає індекс шостого елемента в масиві ext. Подібним чином використовуються масиви покажчиків для зберігання повідомлень про можливі помилки [4]:

1. `char *errors[] = {"Cannot open file",
"Cannot`
2. `close file", "Allocation error", "System
error" }; //.`

При такому оголошенні рядкові константи розташовуються в пам'яті в розділі констант, масив покажчиків складається з чотирьох елементів, для яких теж виділяється пам'ять. Значеннями цих елементів будуть адреси, що вказують на початок строкових констант.

3.1.3 Клас string бібліотеки C ++

Стандартна бібліотека мови C ++ містить опис класу string, який надає багатий набір можливостей для роботи з рядками. Об'єкт класу string інкапсулює рядок у вигляді масиву символів, а також володіє вичерпним набором методів для роботи з рядком. Для використання класу необхідно включити в текст програми заголовки

```
#include <string>
```

після чого можна створювати об'єкт класу string

```
string stroka ( "Це рядок \n");
```

або за допомогою покажчика

```
string * stroka = new string ( "Це рядок \n"); //.
```

Тут конструктору класу передається строкова константа для початкової ініціалізації рядка. Якщо початкова ініціалізація відсутня, то створюється порожній рядок. Рядку можна привласнювати значення рядка

```
stroka = "Інша рядок \ n"; //.
```

Щоб вивести рядок на екран можна відправити її в потік cout

```
cout << stroka; //.
```

Однак вивести її за допомогою функції printf можна тільки після перетворення в стандартний масив символів, що закінчується нулем

```
printf ( "Виводимо рядок% s", stroka.c_str ()); //.
```

Тут метод c_str () повертає потрібний масив символів. Доступ до окремих елементів рядка символів можливий через операцію взяття індексу, наприклад,

```
stroka [5] = '!'; //.
```

По суті всі дії з рядком реалізовані у вигляді методів класу string. Тому перерахуємо імена і призначення основних методів цього класу:

```
begin () повертає покажчик на перший символ рядка,
end () повертає покажчик на уявний символ, наступний за останнім,
append (рядок) додає символ або рядок в кінець рядка,
at (номер) повертає символ, що знаходиться у зазначеній позиції,
c_str () повертає рядок у вигляді масиву символів з завершальним нулем,
clear () видаляє всі символи рядка, залишаючи її порожній,
compare (рядок) порівнює два рядки,
data () повертає покажчик на перший символ рядка,
empty () перевіряє рядок на порожнечу,
erase () видаляє символи із зазначених позицій,
find (підрядок) повертає номер позиції першої появи підрядка в рядку,
find_first_not_of (рядок) знаходить номер позиції першого символу в рядку, що не входить в набір символів іншого рядка,
find_first_of (рядок) знаходить номер позиції першого символу в рядку, що входить в набір символів іншого рядка,
find_last_not_of (рядок) знаходить номер позиції останнього символу в рядку, що не входить в набір символів іншого рядка,
find_last_of (рядок) знаходить номер позиції останнього символу в рядку, що входить в набір символів іншого рядка,
getline зчитує рядок з потоку введення,
insert (номер, підстрока) вставляє підрядок в зазначену позицію рядка,
length повертає довжину рядка
max_size повертає максимально можливий розмір рядка,
rfind здійснює пошук останньої появи підрядка в рядку,
```

size повертає довжину рядка.

На окрему увагу заслуговує функція `replace` (не плутайте з методами). Вона дозволяє замінити один або кілька символів рядка іншими символами. Як правило вона використовується в наступному контексті

```
replace (stroka.begin (), stroka.end (), '5', '*'); //
```

Тут символ '5' змінюється на зірочку у всьому рядку.

3.2 Робота з файлами

3.2.1 Відкриття файлів

Програмування взаємодії з файлами необхідно майже у всіх практично значущих програмах. Основні функції для роботи з файлами описані в бібліотеці `stdio` і являють собою операції відкриття файлу, читання або запису в файл, закриття файлу. Для доступу до файлу застосовується тип даних `FILE`, покажчик на який буде використовуватися для зв'язку з файлом і служити так званої "файлової змінної". Таким чином, спочатку необхідно описати покажчик `f` на змінну типу `FILE`.

```
FILE * f; //
```

Далі потрібно відкрити файл. Для цього використовується функція `fopen` (). В результаті своєї роботи функція `fopen` () динамічно створює в пам'яті змінну типу `FILE` і повертає її адресу. Аргументами для `fopen` () служать два рядки. Перша шлях до файлу, побудований за правилами операційної системи. Другий рядок містить символи, що відображають режим відкриття файлу. Символ 'r' (read) означає, що файл відкривається для читання. Символ 'w' (write) означає відкриття файлу для перезапису. Старе вміст файлу безслідно губиться, а в разі відсутності файлу він створюється. Символ 'a' (append) означає відкриття файлу для додавання. Якщо файл не існує, він створюється. Символи 't' і 'b' використовуються для відкриття текстових і бінарних файлів відповідно. У наступному прикладі виконується відкриття трьох файлів.

```
FILE * f1, * f2, * f3;
f1 = fopen ( "temp.txt", "rt");
f2 = fopen ( "z: /Prog/file.cpp", "w");
f3 = fopen ( "z: \\ Prog \\ file.cpp", "at");
```

Текстовий файл temp.txt відкривається для читання і передбачається, що він знаходиться в одній папці з програмою. Файл file.cpp відкривається двома способами. У першому з них для поділу папок в дорозі використовується прямий слеш /, а в другому подвоєний зворотний слеш (згідно з правилами мови C).

При невдалій спробі відкриття файлу, наприклад, при відкритті на читання неіснуючого файлу, функція fopen () повертає нульовий покажчик NULL. Тому після команди відкриття файлу доцільно перевірити отриманий покажчик:

```
1. #include <stdio.h>
2. ...
3. FILE *f = fopen("file.txt", "rt"); if (f ==
   NULL)
4. { printf("Ошибка открытия файла!!!"); }
5. ...
```

Крім Власний повідомлення про помилку можна скористатися також системним повідомленням, в якому більш точно вказані причини помилки. Для цього замість printf () необхідно скористатися функцією perror ()
perror("Ошибка открытия файла!!!"); //.

Спочатку на екран виведеться повідомлення "Помилка відкриття файлу !!!", а потім системне повідомлення про помилку.

3.2.2 Текстові та бінарні файли

З точки зору програміста файл являє собою потенційно нескінченну стрічку записів будь-якого розміру. Однак видно не всі записи одночасно, а тільки їх частина, як ніби дивимосся на стрічку через уявне вікно, зване покажчиком (не плутати з покажчиком FILE *). У мові C прийнято вважати, що розмір вікна дорівнює одному байту. При зчитуванні або запису вказівник переміщується по стрічці. В кінці файлу знаходиться так званий "ознака кінця файлу", що позначається в мові C константою EOF.

Числові дані, якими оперує програма зберігаються в пам'яті ЕОМ з тому форматі, який передбачений типом цих даних. Наприклад, для змінної цілого типу int виділяються 4 байта. Дані

зберігаються в додатковому коді. При виведенні значення змінної на екран вона перекладається в текстовий, зрозумілий людині вид, тобто в послідовність символів. Точно також в файлі значення змінної може зберігатися або в двійковому вигляді (як в пам'яті ЕОМ), або в текстовому як послідовність символів. У зв'язку з цим файли поділяються на виконавчі (бінарні) і текстові. При роботі з бінарними файлами операції читання і запису не вимагають ніякого перетворення даних, а при роботі з текстовими файлами потрібно перетворювати дані з текстового або в текстовий вигляд.

3.2.3 Читання з файлу

Читання бінарних файлів.

Читання бінарних даних з файлу виробляє функція `fread()`, яка має 4 аргументу: ім'я масиву, в який буде проводитися читання, розмір одного елемента масиву, число елементів для читання, покажчик файлової змінної. Що повертається функцією `fread()` значення дорівнює реальній кількості лічених елементів. Нехай, наприклад, необхідно вважати з файлу 512 дійсних чисел типу `float`. Передбачимо масив `mas`, в який поміщаються дані, змінну `kolvo` для результату зчитування і покажчик на файлову змінну

```
unsigned int kolvo;
float mas [512];
FILE * f; //.
```

Після відкриття файлу Виконяемие зчитування

```
kolvo = fread (mas, sizeof (float), 512, f);
```

Тут функція `sizeof (float)` обчислює розмір зчитує елемента, а змінна `kolvo` приймає значення, рівне реальній кількості лічених елементів. Її значення може бути менше передбачених 512 елементів, оскільки несподівано може зустрітися ознака кінця файлу.

Читання текстових файлів.

Читання текстових файлів можна здійснити за допомогою функції форматного введення `fscanf()`. Вона перетворює лічені дані з текстового подання в бінарне. Використання функції `fscanf()` аналогічно функції введення з клавіатури `scanf()`, тільки першим аргументом повинен бути покажчик на файлову змінну

```
scanf (Показчик на файл, "Керуюча рядок", Arg1,
Arg2, ...); //.
```

Ось кілька прикладів використання функції `fscanf ()`:

```
int K, kolvo;
float X, Y;
char C;
char S [80];
FILE * f;

. . .
fscanf (f, "% d", & K);
fscanf (f, "% f", & X);
fscanf (f, "% c", & C);
fscanf (f, "% s", S);
fscanf (f, "% f% f", & X, & Y);

. . .
```

Змінна `kolvo` приймає значення, рівне кількості введених даних.

3.2.4 Запис в файл

Запис бінарних даних.

Запис бінарних даних в файл можна здійснити за допомогою функції `fwrite ()`. Вона має 4 аргументу: ім'я масиву, значення елементів якого необхідно записати в файл, розмір одного елемента масиву, число записуваних елементів, показчик файлової змінної. Що повертається функцією `fwrite ()` значення дорівнює реальній кількості записаних елементів. Нехай, наприклад, необхідно записати в файл 512 дійсних чисел типу `float` з масиву `mas`

```
unsigned int kolvo;
float mas [512];
FILE * f; //.
```

Після відкриття файлу виконуємо запис

```
kolvo = fwrite (mas, sizeof (float), 512, f); //.
```

Тут функція `sizeof (float)` обчислює розмір записуваного елемента, а змінна `kolvo` приймає значення, рівне реальній кількості

записаних елементів. Її значення може бути менше передбачених 512 елементів, оскільки несподівано може закінчитися місце на диску.

Запис в текстовий файл.

Запис бінарних даних в текстовий файл можна здійснити за допомогою функції форматного виведення `fprintf()`. Вона перетворює записуються дані в текстове представлення у відповідність з форматної рядком. Використання функції `fprintf()` аналогічно функції виведення на екран `printf()`, тільки першим аргументом повинен бути покажчик на файлову змінну.

```
fprintf (Покажчик на файл, "Керуюча рядок", Арг1,
Арг2, ...); //.
```

Керуюча рядок може містити компоненти трьох типів:

- звичайні символи, які просто виводяться на екран;
- керуючі константи з табл. 5;
- специфікації перетворення.

Кожна специфікація перетворення служить для виведення на екран чергового даного зі списку аргументів `Арг1`, `Арг2`, ... Починається специфікація знаком%, закінчується символом перетворення. У наступному прикладі:

```
fprintf (f, "Результат обчислень дорівнює% 12.3f\n", y);
```

символом перетворення служить буква `f`. Отже, функція `printf()` виведе на екран замість% 12.3f значення змінної `y`, яка стоїть в списку аргументів. При цьому всього для значення змінної `y` буде передбачено 12 знаків, серед них 3 для дробової частини.

3.2.5 Закриття файлу

Після закінчення роботи з файлом його необхідно закрити. Для закриття файлу використовується функція `fclose()`, аргументом якої є покажчик на файлову змінну, наприклад `fclose(f); //`.

При вдалому закритті функція повертає нуль, а при помилці значення `-1`. Тому при закритті файлу можна використовувати наступного виду перевірку

```
if (fclose (f) <0)
{
```

```
printf ( "Неможливо закрити файл !!!"); } //.
```

3.2.6 Позиціонування покажчика файлу

Нерідко зустрічається необхідність зчитувати і записувати дані в файл непослідовно. При читанні або записі кількох байт покажчик поточної позиції автоматично зміщується на потрібне число позицій. Існує також функція примусового позиціонування покажчика поточної позиції в потрібне місце `fseek()`. Ця функція має три аргументи: покажчик на файлову змінну `FILE * f`, величина зсуву цілого знакового типу, величина цілого типу, яка характеризує точку відліку зміщення. Для третього аргументу передбачені три можливих значення, задані як цілі константи: `SEEK_CUR = 1` - зміщення відраховується від поточної позиції вказівника, `SEEK_SET = 0` - зміщення відраховується від початку файлу, `SEEK_END = 2` - зміщення відраховується від кінця файлу. Розглянемо кілька прикладів:

```
fseek(f, 0, SEEK_SET); fseek(f, -8, SEEK_END);
fseek(f, 5, SEEK_CUR); //.
```

Перша з команд позиціонує покажчик в початок файлу, друга в восьми байтах від кінця файлу, третя команда зміщує покажчик на 5 байт вперед.

Функція `ftell()` дозволяє дізнатися номер поточної позиції вказівника, наприклад,
`N = ftell(f); //`

Функція `feof()` дає можливість перевірити, чи досяг покажчик кінця файлу. Вона повертає нульове значення, якщо кінець файлу досягнутий.

3.2.7 Поняття потоку введення-виведення

Всі сучасні операційні системи підтримують так званий механізм потоків. Потік являє собою сукупність електронних даних. Він завжди має тільки один напрямок і по відношенню до програми є потоком або введення, або виведення. Програма може читати дані з потоку введення і виводити дані в потік виводу. По суті робота з

потокom нічим не відрізняється від роботи з файлом. Тому для реалізації потоків в мові C використовуються принципи взаємодії з файлами. Доступ до потоку здійснюється за допомогою змінної типу FILE * (показчик на файлову змінну). Існують три стандартних потоки, що підключаються до кожної програми: * stdin вхідний потік, пов'язаний з клавіатурою; * Stdout вихідний потік, пов'язаний з екраном; * Stderr вихідний потік для виведення повідомлень про помилки, теж пов'язаний з екраном. Таким чином, наступні команди виведення повідомлення на екран рівносильні

```
printf ( "Повідомлення");  
fprintf (stdout, "Повідомлення"); //.
```

3.2.8 Функції бібліотеки введення-виведення

Крім перерахованих вище бібліотека введення-виведення stdio містить ще багато корисних функцій. Далі в якості довідкової інформації наведені прототипи деяких з них.

Функція remove видаляє файл з ім'ям filename

```
int remove (char * filename); //.
```

Функція rename перейменовує файл old в файл new

```
int rename (char * old, char * new);
```

табл. 3.1 наведені функції введення-виведення бінарних даних з коротким описом і прототипом.

3.2.9 Класи і об'єкти потоків введення-виведення

C ++ застосовується об'єктно-орієнтований підхід до реалізації обміну даними з потоками введення-виведення. Стандартна бібліотека C ++ iostream містить опису класів, інкапсулюючих потоки введення-виведення. Базовим для класів потоків є клас ios. Його спадкоємці класи istream і ostream відповідають за введення і виведення відповідно. Вони мають загального спадкоємця клас iostream. При запуску програми автоматично створюються об'єкти cin класу istream і об'єкти cout, cerr, clog класу ostream. Перший з них пов'язаний з клавіатурою і підтримує операцію взяття з потоку >>. Решта пов'язані з екраном і підтримують операцію приміщення в потік <<.

Для організації файлового введення-виведення класи `istream`, `ostream` і `iostream` мають спадкоємцями `ifstream`, `ofstream` і `fstream`

Таблиця 3.1 – Функції введення-виведення бінарних даних

	Для будь-якого потоку
<code>fgetc</code>	Зчитує один символ з потоку A, <code>int fgetc (FILE * A)</code>
<code>fgets</code>	Зчитує рядок s довжиною не більше n символів з потоку <code>A, char * fgets (char * s, int n, FILE * A)</code>
<code>fputc</code>	Виводить символ C в потік A, <code>int fputc (int C, FILE * A)</code>
<code>fputs</code>	Виводить рядок символів S в потік A, <code>int fputs (char * S, FILE * A)</code>
<code>getc</code>	Зчитує з потоку A один символ, <code>int getc (FILE * A)</code>
<code>putc</code>	Виводить символ C в потік A, <code>int putc (int C, FILE * A)</code>
<code>putw</code>	Виводить в потік A ціле значення W, <code>int putw (int W, FILE * A)</code>
<code>getw</code>	Зчитує з потоку A ціле число, <code>int getw (FILE * A)</code>
	Для потоку <code>stdin</code>
<code>gets</code>	Зчитує рядок символів S з потоку <code>stdin</code> , <code>char * Gets (char * S)</code>
<code>getchar</code>	Зчитує символ з потоку <code>stdin</code> , <code>int getchar (void)</code>
	Для потоку <code>stdout</code>
<code>putchar</code>	Виводить символ C в потік <code>stdout</code> , <code>int putchar (int C)</code>
<code>puts</code>	Виводить рядок S в потік <code>stdout</code> , <code>int puts (const char * S)</code>

відповідно. Перераховані класи мають безліч методів для управління введенням і виведенням і можуть використовувати так звані маніпулятори. Наприклад, цілі числа зазвичай інтерпретуються як десяткові. Помістивши в потік `cout` маніпулятор `hex`, отримаємо на екрані числа в шістнадцятковій системі

```
cout << "N =" << hex << N << endl; //.
```

Аналогічно працюють маніпулятори `oct` і `dec`. Точність виведення в потік речових чисел регулює маніпулятор `precision (m)`, де `m` кількість знаків після коми. При `m = 0` встановлюється точність за замовчуванням (шість знаків).

Використання файлового введення-виведення стає доступним після включення в програму заголовки `fstream`. Щоб відкрити файл на читання необхідно створити об'єкт класу `ifstream` і передати ім'я файлу його конструктору як параметр

```
ifstream fileIn ( "d: \\ text \\ fileIn.txt"); //.
```

Аналогічно виконується відкриття файлу для запису, тільки використовується об'єкт класу `ofstream`

```
ofstream fileOut ( "d: \\ text \\ fileOut.txt");  
//.
```

Імена об'єктів `fileIn` і `fileOut` можна використовувати поряд зі стандартними потоками `cin` і `cout`, наприклад,

```
for (k = 1; k <= 100; k ++ ) {fileOut << x [k];}  
//.
```

Уточнити, як саме слід відкривати файл, можна за допомогою другого (необов'язкового) аргументу конструктора. Для цього можна використовувати такі константні значення `ios :: app` відкриття файлу на додавання, `ios :: ate` - на додавання з можливістю переміщатися по файлу, `ios :: trunc` на перезапис (минуле вміст втрачається), `ios :: nocreate` якщо файл не існує, то відкриття не виконується, `ios :: noreplace` якщо файл існує, то відкриття не виконується. Так, наприклад, для відкриття файлу на додавання необхідно писати

```
ifstream fileOut ( "d: \\ text \\ fileOut.txt",  
ios :: app);
```

Після використання файл необхідно закрити за допомогою методу `close ()`.

3.3 Практичне завдання

При виконанні практичного завдання дотримуйтесь наступної послідовності дій.

- 1)Обов'язково вивчіть опис, пропонуване вище.
- 2)Розберіть приклади, що наводяться в описі.

- 3) Осмисліть завдання, запропоновану в вашому варіанті.
- 4) Складіть словесний алгоритм вирішення задачі.
- 5) Уявіть алгоритм рішення у вигляді блок-схеми.
- 6) Переведіть алгоритм на мову програмування.

3.3.1 Варіанти завдань

Варіант 1.

Напишіть функцію наближеного обчислення квадратного кореня за формулою $x_n = x_{n-1} / 2 + a / 2x_{n-1}$, $x_1 = 1$, $x \rightarrow \sqrt{a}$. Знайдіть корені всіх чисел з вхідного текстового файлу, додати їх у той же файл поруч з вихідними.

Варіант 2.

Дано речення, слова в ньому розділені одним або декількома пропусками. Пропозиція записано в текстовому файлі. Замініть кожне слово кількістю букв 'а' в ньому. Використовуйте функцію підрахунку кількості заданої букви в слові.

Варіанти 3-18.

У текстовому файлі знаходяться записи про номери телефонів. У кожному рядку записана інформація про один абонента: № телефону, ПІБ, адреса. Завдання необхідно вирішити з використанням процедур та функцій.

- 3) Напишіть програму, яка здійснює додавання запису в файл з перевіркою коректності даних.
- 4) Напишіть програму, яка видаляє з файлу зазначену запис.
- 5) Напишіть програму, яка згідно із запитом користувача виводить інформацію про абонентів, прізвище яких починається з вказаної букви.
- 6) Напишіть програму, яка згідно із запитом користувача виводить інформацію про абонентів, підключених до однієї АТС.
- 7) Напишіть програму, яка здійснює пошук абонента із заданим прізвищем.
- 8) Напишіть програму, яка здійснює пошук абонента із заданим ім'ям.
- 9) Напишіть програму, яка здійснює пошук абонента із заданим номером.

- 10) Напишіть програму, яка для кожної АТС виводить відсоток заповнювання телефонних номерів.
- 11) Напишіть програму, яка виводить діапазони вільних номерів телефонів для кожної АТС.
- 12) Напишіть програму, яка сортує записи в файлі в порядку зростання номера телефону.
- 13) Напишіть програму, яка сортує записи в файлі в алфавітному порядку прізвищ.
- 14) Напишіть програму, яка згідно із запитом користувача виводить інформацію про абонентів, які проживають на вказаній вулиці.
- 15) Напишіть програму, яка згідно із запитом користувача виводить інформацію про абонентів, які проживають в зазначеному будинку.
- 16) Напишіть програму, яка згідно із запитом користувача виводить інформацію про абонентів, з зазначеними прізвищем і ім'ям.
- 17) Напишіть програму, яка згідно із запитом користувача виводить інформацію про абонентів, прізвище яких закінчується зазначеної буквою.
- 18) Напишіть програму, яка виводить діапазони зайнятих номерів телефонів для кожної АТС без вказівки інформації про абонентів.

Варіанти 19-27.

У текстовому файлі знаходяться записи про книги в бібліотеці. У кожному рядку записана інформація про одну книжку: ПІБ автора, назву, рік видання, кількість сторінок. Завдання необхідно вирішити з використанням процедур функцій.

- 19) Напишіть програму, яка здійснює додавання запису в файл з перевіркою коректності даних (чи не існує вже додається запис).
- 20) Напишіть програму, яка видаляє з файлу вказаний запис.
- 21) Напишіть програму, яка згідно із запитом користувача виводить інформацію про книгах зазначеного автора.
- 22) Напишіть програму, яка згідно із запитом користувача шукає книгу по зазначеній частині назви.
- 23) Напишіть програму, яка сортує записи в файлі в алфавітному порядку прізвищ авторів.
- 24) Напишіть програму, яка згідно із запитом користувача виводить інформацію про книги, випущених після зазначеного року.
- 25) Напишіть програму, яка згідно із запитом користувача виводить інформацію про книгах більше заданого числа сторінок.

26) Напишіть програму, яка згідно із запитом користувача виводить інформацію про книги, прізвище одного з авторів яких закінчується вказаною буквою.

27) Напишіть програму, яка обчислює, в який рік було випущено максимальну кількість книг.

Додаток А
Приклад оформлення титульної сторінки

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

Кафедра МiНЕ

ЛАБОРАТОРНА РОБОТА
з дисципліни
„ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ“

(тема роботи)

Виконав:
студент ____ курсу, група РТ-____
(Ініціали, Прізвище) _____

Прийняв:

(посада) _____

(Ініціали, Прізвище)

20 __ р.

Додаток Б

Підпрограми. Вказівники

Поняття підпрограм. Функції

У практиці програмування часто є необхідність виконувати одні і ті ж обчислення, але при різних вихідних даних. Для виключення повторень однакового коду і спрощення програми повторювані обчислення виділяють в самостійну частину програми, яка може бути використана багаторазово в міру необхідності. Така автономна частина програми, що реалізує певний алгоритм, оформлена у вигляді окремої синтаксичної конструкції, забезпечена ім'ям і допускає звернення до неї з різних частин програми, називається підпрограмою. Іншими словами підпрограма це послідовність операторів, які визначені й записані тільки в одному місці програми, проте їх можна викликати для виконання з однієї або декількох точок програми. Кожна підпрограма визначається унікальним ім'ям.

Активне використання підпрограм обумовлено так-же тією обставиною, що алгоритм вирішення задачі часто проектується шляхом декомпозиції всієї завдання на окремі під задачі. Зазвичай під задачі оформляються як підпрограми . У мові C ++ підпрограми реалізуються у вигляді функцій. Функція це іменована послідовність описів і операторів. Функція може розташовуватися як у файлі основної програми, так і в окремому файлі. За визначенням функція повинна повертати викликає її програмі результат своєї роботи. Однак, якщо за змістом завдання повернення результату не потрібно, то при описі функції використовують службове слово `void` для типу повертається результату.

Опис функцій

Опис функції виконується згідно наступним шаблоном

1. Тип ІмяФункції (список параметрів)
2. {
3. Тіло функції;
4. } //.
- 5.

Тип визначає тип результату, що повертається функцією. Якщо тип не вказано, то вважається, що функція повертає результат типу `int`.

Параметри функції це змінні, за допомогою яких передаються дані з місця виклику функції в її тіло. Список параметрів складається з переліку їх типів і імен, перерахованих через кому. Якщо функція не має параметрів, то їх список може бути відсутнім, але круглі дужки все одно необхідні. Перерахуємо кілька прикладів заголовків функцій

```
1. float diskr(float a, float b, float c) //,  
2. void podprogramma() //, int prog1() //.
```

Передача результату (значення) з функції в зухвалу програму здійснюється за допомогою оператора return. Використання цього оператора дуже різноманітне. Якщо після оператора return знаходиться повертається значення, то виконання функції припиняється і значення передається викликає програмі. Якщо return використовується без значення, що повертається, то його призначення полягає в тому, щоб припинити виконання функції. Функція може містити один, кілька або жодного оператора return. В останньому випадку виконання функції припиняється після виконання останнього її оператора.

Область дії змінних

Область дії (видимості) змінної це правила, які встановлюють, які дані доступні з даного місця програми. У мові С кожна функція це окремий блок програми. Потрапити в тіло функції не можна інакше, як через виклик даної функції.

В точки зору області дії змінних розрізняють три види змінних: глобальні, локальні і формальні параметри. Правила області дії визначають, де кожна з них може застосовуватися.

Локальні змінні це змінні, оголошені всередині блоку, зокрема всередині функції. Мова С підтримує просте правило: змінна може бути оголошена всередині будь-якого блоку програми. Область дії локальної змінної блок. Локальна змінна існує поки виконується блок, в якому ця змінна оголошена. При виході з блоку ця змінна (і її значення) втрачається.

Формальні параметри це змінні, оголошені при описі функції як її аргументи. Функції можуть мати деяку кількість параметрів, які використовуються при виконанні функції для передачі значень аргументів. Формальні параметри можуть використовуватися в тілі

функції так само, як локальні змінні, якими вони по суті справи і є. Область дії формальних параметрів блок, який є тілом функції.

Глобальні змінні це змінні, оголошені поза будь-якої функції. На відміну від локальних глобальні змінні можуть бути використані в будь-якому місці програми нижче їх оголошення. Область дії глобальної змінної вся програма.

Використання глобальних змінних має свої недоліки: вони займають пам'ять протягом усього часу роботи програми, використання глобальних змінних робить функції менш загальними і ускладнює їх використання в інших програмах.

Вказівники

Розуміння і правильне використання покажчиків є основою для створення професійних програм на мові С. Покажчик це змінна, яка призначена для зберігання і використання в програмі адреси деякого об'єкта. Тут мається на увазі адреса в пам'яті комп'ютера. Адреса являє собою просте ціле число, але його не можна трактувати як змінну або константу цілого типу. Якщо змінна за змістом є покажчиком, то вона повинна бути відповідним чином оголошена. Форма оголошення покажчика наступна:

Тип * ім'я_змінної; //.

цьому оголошенні тип деякий допустимий для мови С (С++) тип даних. Це тип тієї змінної, адреса якої буде зберігати покажчик. Знак * означає, що наступна за ним змінна є покажчиком. Наприклад:

1. `char * ch;`
2. `int * temp, i, * j;`
3. `float * pf, f; //.`

Тут оголошені покажчики `ch`, `temp`, `j`, `pf`, змінна і типу `int` і змінна `f` типу `float`.

З покажчиками пов'язані дві спеціальні операції: `&` і `*`. Обидві ці операції є унарними, тобто мають один операнд, перед яким вони ставляться. Операція `&` відповідає за змістом операції взяття (визначення) адреси. Операція `*` є операцією взяття (визначення) значення за вказаною адресою. Дані операції можна сплутати з відповідними їм по написанню бінарними операціями, порозрядним AND і операцією множення, так як вони є унарними, що завжди видно з контексту програми.

При оголошенні покажчика дуже важливим є базовий тип, який повідомляє компілятору скільки байт пам'яті займає змінна, на яку вказує даний покажчик. Найпростіші дії з покажчиками проілюструємо на прикладі наступної програми:

```

1. include <iostream.h>
2. int main (void)
3. {
4. //Объявляем переменные x и y
5. float x= 12.3, y;
6. // Объявляем указатель p
7. float *p;
8. // Присваиваем указателю p адрес переменной
   x
9. p=&x; // Переменной y присваиваем значение
10.     // переменной, адрес которой
11.     // содержит указатель p
12.     y=*p; // Выводим на экран x и y
13.     cout << "x=" << x << "   y=" << y <<
        endl;
14.     //Увеличиваем на 1 значение,
15.     //взятое по указателю p
16.     (*p)++;
17.     //Выводим на экран x и y
18.     cout << "x=" << x << "   y=" << y <<
        endl;
19.     //Добавляем 1 к произведению значения,
        взятого
20.     //по указателю p на y
21.     y=1+(*p)*y; //Выводим на экран x и y
22.     cout << "x=" << x << "           y=" << y <<
        endl;
23.     } // конец программы.

```

До вказівників можна застосовувати операцію присвоювання, якщо вони є вказівниками одного типу, наприклад:

```

1. include <iostream.h>
2. int main(void)
3. {int x=12;

```

```

4. int *p, *g;
5. p=&x;
6. g=p;
7. printf("%p\n",p);
8. printf("%p\n",g);
9. printf("%d\n%d\n",x,*g);
10.    } // кінець програми

```

У цьому прикладі наведена специфікація формату% р функції printf (), яка використовується для виведення адреси пам'яті в шістнадцятковій формі.

Не можна створити змінну типу void, але можна створити покажчик на такий тип. Вказівником на void можна привласнити значення покажчика будь-якого іншого типу. При зворотному привласненні необхідно використовувати явне перетворення покажчика на void.

Наприклад, розглянемо наступний фрагмент:

```

1. void *pv;
2. float f, *pf;
3. pf=&f;
4. pv=pf;
5. pf=(float*)pv; // .

```

Тут вказівник pv приводиться до типу (float *), тобто до покажчика на змінну типу float.

У мові С допустимо привласнити покажчику будь-яку адресу пам'яті. Однак якщо оголошений покажчик на ціле (int * p), а за адресою, що присвоєний даному покажчику, знаходиться змінна x іншого типу, то при компіляції програми буде видане повідомлення про помилку в рядку p = & x. Цю помилку можна виправити, перетворивши адреса змінної x до потрібного типу покажчика явним перетворенням типу: p = (int *) & x; але при цьому втрачається інформація про тип змінної x.

Над вказівниками можна виробляти арифметичні операції: додавання і віднімання. Арифметичні дії над покажчиками мають свої особливості. Розглянемо програму:

```

1. int main(void)

```

```

2. {
3. int *p;
4. int x=12;
5. p=&x;
6. printf("%p\n%p", p, ++p);
7. }  // .

```

При виконанні цієї програми побачимо, що в результаті операції `++ p` значення покажчика `p` збільшиться на 2, а не на 1. Це правильно, тому що в такому значенні покажчика вказує на адресу наступного цілого, а не на наступну адресу (ціле займає 2 байта). Таким чином, за будь-якої операції `++ p` значення покажчика буде збільшуватися на кількість байт, займаних змінної базового типу покажчика.

До вказівників можна додавати або віднімати деяке ціле. Нехай покажчик `p` має значення 4000 і вказує на ціле. Тоді в результаті виконання оператора `p = p + 5`; значення покажчика стане рівним 4010. Загальна формула для обчислення значення покажчика після виконання операції `p = p + n`; матиме вигляд:

$p = p + n * \text{Размер_в_байтах_переменной_базового_типа.}$

Аналогічні правила віднімання цілих констант з значення покажчика. Можна також вчитати один покажчик з іншого. Інші арифметичні операції над покажчиками заборонені, тобто не можна складати два покажчика, множити один на одного, ділити.

Вказівники можна порівнювати. Чи можливо застосувати всі 6 операцій порівняння. Порівняння `p < g` дає "істину" якщо адреса, що знаходиться в `p`, менше адреси, що знаходиться в `g`.

У мові C допускається використання вказівників, що вказують на покажчик. У цьому випадку опис має наступний вигляд:

```
int ** point; //.
```

Тут `point` має тип покажчик на покажчик `int`. Відповідно, щоб отримати цілочисельне значення змінної, на яку вказує `point`, треба в вираженні використовувати `** point`. Розглянемо приклад:

```

1. #include <stdio.h>
2. int main(void)
3. {
4. int i;

```

```

5. int *pi;
6. int **ppi;
7. i=12;
8. pi=&i;
9. ppi=&pi;
10.    printf("i = %d pi = %p ppi = %p
11.    **ppi = %d\n", i, pi, ppi, **ppi);
12.    }  // .

```

Після того як покажчик був оголошений, але до того як йому було присвоєно значення, покажчик містить невизначений значення. Спроба використовувати таке значення може викликати помилку при виконанні програми і навіть порушити роботу операційної системи. Всім вказівниками, не використовуваним в даний момент доцільно привласнювати зумовлене значення NULL. Воно є в деякому сенсі заземленням покажчика.

Передача параметра за посиланням

У мові С всі аргументи функції передаються за значенням. Це означає, що при виконанні функції в системний стек поміщаються фактичні значення аргументів функції. У функції створюються локальні копії аргументів, значення яких зчитуються з стека. Далі функція використовує ці локальні змінні і може змінювати їх. При виході з функції локальні змінні знищуються, змінені значення параметрів губляться. Таким чином, в мові С викликана функція не може змінити значення змінних, вказаних в якості фактичних параметрів при зверненні до неї. Наприклад, функція `swap ()`, яка повинна змінювати значення параметрів місцями, не буде цього робити:

```

void swap(int a, int b)
{
int tmp=a; a=b; b=tmp;
} // .

```

Але тим не менше функцію можна пристосувати для зміни аргументів. Для цього необхідно в якості параметра передавати адресу змінної, яку треба змінювати, тобто передавати покажчик на змінну.

Такий прийом в мові C називається передачею параметра через покажчик. Викликана функція повинна описувати аргумент як покажчик і звертатися до фактичної змінної опосередковано, через розіменування покажчика. Тепер функцію `swap ()` можна описати таким чином:

```
void swap(int *a, int *b)
{
    int tmp=*a; *a = *b; *b=tmp; } //.
```

Тоді для обміну місцями значень змінних `x` і `y` необхідно використовувати функцію `swap (& x, & y)`.

Ще одним способом непрямої передачі параметра функції є передача за посиланням. Тоді аргументами функції `swap ()` будуть посилання на інші змінні

```
void swap (int & a, int & b)
{
    int tmp = a; a = b; b = tmp;
} //.
```

При зверненні до функції в якості фактичних параметрів необхідно вказувати імена змінних `x` і `y`: `swap (x, y)`.

Динамічне виділення пам'яті

Крім розглянутих елементарних дій з вказівниками з їх допомогою можна також створювати нові змінні і виділяти пам'ять динамічно в ході роботи програми. Для цього призначений оператор `new`:

указатель = new тип;
Наприклад:

```
int *p; // указатель на целое
p = new int; // выделяем память.
```

Після операції `new` вказується тип даних, які будуть зберігатися в виділеній області пам'яті, адреса початку виділеної області присвоюється покажчику. Після цього можна користуватися виділеними осередками, наприклад

```
*p = 155; // присвоили значение.
```

Коли динамічно виділена пам'ять більше не потрібна її можна звільнити за допомогою оператора delete:

```
delete покажчик;
```

наприклад,

```
delete p; // звільнили пам'ять.
```

Видалення покажчика при цьому не відбувається, пам'ять, адреса якої знаходиться в покажчику, оголошується вільною

Рекурсивні функції

Рекурсивна функція це функція, в тілі якої є виклик самої себе. Використання рекурсивних функцій буває зручним при програмуванні ряду завдань, наприклад обчисленні факторіала деякого числа N або швидкого сортування масиву.

```
int factorial (int n)
{
    int a;
    if (n == 1) return 1; // передбачили вихід a =
    factorial (n-1) * n; // виклик себе return a;
} //.
```

Виклик функції при рекурсії НЕ створює нову копію функції в пам'яті, а створює нові копії локальних змінних і параметрів. З рекурсивної функції необхідно передбачити вихід, щоб не спровокувати зависання обчислювальної системи. Необхідно пам'ятати, що при великій кількості рекурсивних викликів буде відбуватися швидке заповнення стека, розмір якого обмежений. Це може викликати зупинку програми. Тому, використання рекурсії як методу програмування, має бути обережним.

Додаток В

Особливості мультіфайлового програмування

Компіляція і компоновка

Як правило текст серйозної програми незручно розміщувати в одному файлі. У такій програмі незручно розбиратися, її важко модифікувати. Доцільно розбивати текст програми на окремі функціонально завершені блоки і розміщувати їх в різних файлах. Потім файли з вихідними кодами блоків компілюються окремо, і тільки потім отримані об'єктні коди компонуються в єдиний виконуваний файл. Розглянемо цей процес докладніше. Створимо каталог проекту. Розділимо текст програми обчислення доходу по вкладу, наведеної на стор. 34 на три файли. У першому з них func.cpp розмістимо функцію обчислення доходу

```
1. float fun_dohod (float sum, float srok,
   float stavka) {
2. return (sum * stavka / 100) * srok / 365; }
   //.
```

Другий файл func.h заголовки, в ньому знаходиться прототип функції fun_dohod.

Прототипом функції називається її заголовна рядок із зазначенням імені функції і типів аргументів і результату. Третій файл main.cpp містить головну функцію main, в якій використовується функція fun_dohod для обчислення доходу по вкладу.

```
1. #include <iostream.h>
2. #include "func.h"
3. float sum,      //сумма вклада
4. srok,          //срок вклада
5. stavka,        //процентная ставка
6. dohod;         //доход по вкладу
7. int main(void)
8. {
9. cout << "Вычисление дохода по вкладу\n";
10.      cout << "Введите исходные данные:\n";
```

```

11.     cout << "Величина вклада (грн) ->";
12.     cin >> sum;
13.     cout << "\n";
14.     cout << "Срок вклада (дней) ->";
15.     cin >> srok;
16.     cout << "\n";
17.     cout << "Процентная ставка ->";
18.     cin >> stavka;
19.     cout << "\n";
20.     dohod=fun_dohod(sum, stavka, srok);
21.     sum=sum+dohod;
22.     cout << "\n";
23.     cout << "-----\n";
24.     cout << "Доход:" << dohod << "грн\n";
25.     cout << "Конечная сумма:" << sum <<
        "грн\n"; } //.
```

Заголовки використовується для підключення функції `fun_dohod` з файлу `func.cpp` до головної програми за допомогою рядка `#include "func.h"`. Для ілюстрації всього процесу отримання виконуваного файлу виконаємо компіляцію двох файлів з вихідним текстом програми командами:

```
g++ -c func.cpp и
g++ -c main.cpp.
```

В результаті в каталозі проекту з'являться два файли з об'єктним кодом `func.o` і `main.o`. Зазирнути всередину об'єктного коду допомагає утиліта `nm`. команда

```
nm func.o
```

виводить на екран так званий список символів програми, в якому згадується ім'я функції `fun_dohod`.

```
...
```

```
00000000 T __Z9fun_dohodfff
```

Зауважимо, що навпроти всіх символів файлу `func.o` знаходяться нулі. Це означає, що символам не поставлений у відповідність реальну адресу або зсув щодо деякої адреси. команда

```
nm main.o
```

виводить список символів об'єктного коду main.o.

...

U __Z9fun_dohodfff

...

000000fe T _main

00000004 B _srok

00000008 B _stavka

00000000 B _sum

Тут для функції main вказано значення 000000fe, а навпроти рядка з ім'ям функції fun_dohod адреса не вказана, що свідчить про необхідність скріплення імен допомогою таблиць символів. В процесі компонування вираховуються адреси, відповідні іменам символи або підставляються в потрібні місця таблиці, а також підключаються функції з використовуваних стандартних бібліотек. висновок команди nm a.exe

надзвичайно довгий, проте в ньому присутні рядки

...

00401640 T __Z9fun_dohodfff

...

004013ee T _main

...

зіставляють описані в програмі функції з реальними адресами або зсувами адрес.

Автоматична збірка проекту

Згідно з попереднім пунктом для створення виконуваного файлу з двох файлів з вихідним текстом програми потрібно вводити три команди: дві для компіляції кожного файлу і одну для компонування. Припустимо тепер, що файлів з вихідним текстом досить багато, причому функції, описані в одних використовують функції, описані в інших файлах. Тоді зміна текста програми при налагодженні призведе до необхідності перекомпіляції багатьох файлів. Для автоматизації процесу компіляції існує утиліта make. Її робота відбувається за заздалегідь підготовленим сценарієм, розташованому в спеціальному файлі Makefile в каталозі проекту. Вміст Makefile складається з елементів трьох типів: коментарів, макроозначень і цільових зв'язок. Коментарем вважається рядок, що починається знаком дієз. Макроозначення дозволяють призначити ім'я будь-якої рядку, а потім

підставити цей рядок в необхідному місці. Цільові зв'язки є обов'язковим вмістом файлу сценарію і в свою чергу складаються теж з трьох елементів: мета, залежно та правила. У зв'язках вказується, що потрібно зробити для компіляції, що для цього потрібно і як (за допомогою якої команди) це зробити.

Мета це ідентифікатор (будь-яке ім'я або "чарівне слово") з двокрапкою. Залежно це список файлів або інших цілей, розділених пропуском, необхідних для успішного здійснення мети. Правила команди, які необхідно виконати для досягнення мети. Для розглянутого вище двох файлового проекту можна використовувати наступний Makefile.

Приклад файлу сценарію Makefile

```
prog: main.o func.o
    g++ -o prog.exe main.o func.o
main.o: main.cpp
    g++ -c main.cpp
func.o: func.cpp
    g++ -c func.cpp
clean:
    rm -f *.o prog.exe
```

Тут описані чотири цілі: prog, main.o, func.o, clean.

Перша мета prog служить для компонування об'єктних кодів і отримання виконуваного файлу prog.exe. Вона залежить від наявності файлів з об'єктними кодами main.o і func.o, які в свою чергу є цілями. Для досягнення мети prog використовується правило (команда)

```
g++ -o prog.exe main.o func.o.
```

Всі правила обов'язково починаються з нового рядка і з символу табуляції. Цілі main.o і func.o оформлені однаково. Їх залежностями служать файли з вихідними текстами програми, а правила здійснюють компіляцію з придушенням компонування. Нарешті, остання мета clean служить для очищення проекту від результатів компіляції. Вона не має залежностей і її правило являє собою команду видалення файлів з розширенням про і виконуваного файлу prog.exe. Команда видалення запозичена з Linux. Реалізація утиліти make для Windows користується вбудованим емулятором терміналу Linux, тому теж використовує команду видалення rm.

Після написання Makefile для збірки проекту досить запустити утиліту make і вказати їй мета, наприклад

make prog
або
make clean.

Запуск make без параметрів приводить до збірки першої мети, яка за замовчуванням вважається головною.

Сучасні інтегровані середовища програмування містять також утиліти, котрі аналізують вихідний текст програми, залежності між файлами і самі складають Makefile. Саме тому в пунктах 1.5.1 і 1.5.2 згадувалося про створення цілей і файлу сценарію Makefile.

Додаток Г

Елементи об'єктно-орієнтованого програмування

Структури

Структури є розвитком поняття масив. Масиви являють собою об'єднання однотипних змінних. Однак найчастіше логіка програми вимагає об'єднання різнотипних змінних. Нехай, наприклад, програма оперує описами радіоелектронних компонентів, для визначеності інтегральних схем. Кожна з них має характеристики різних типів: маркування, кількість контактів, тип корпусу, напруга живлення, споживана потужність і т.д. Можна об'єднати ці різнотипні характеристики в єдину змінну структуру.

Структура це об'єднання різнотипних змінних. Загальний вигляд опису структури:

```
1. struct ІмяСтруктури {  
2. Тип1 імя1;  
3. Тип2 імя2;  
4. ...  
5. }; //.
```

Змінні, які об'єднані структурою, називаються її полями. Це можуть бути звичайні змінні, масиви, покажчики, інші структури, об'єднання. Опишемо структуру для зберігання даних про студента

```
1. struct student {  
2. char fio [30];  
3. int kurs;  
4. char group [7];  
5. }; //.
```

Оголошення структури є оператором, і тому після такого оголошення повинна стояти крапка з комою. Особливо відзначимо, що оголошення структури не призводить до виділення пам'яті під змінну. Це лише опис каркаса майбутньої змінної, її шаблон. З точки зору компілятора оголошення структури є описом нового користувацького типу даних. Для того щоб оголосити змінні типу student, можна записати

```
student stud1, stud2; //.
```

Тут оголошені дві змінні stud1 і stud2. Під кожен з змінних типу структури виділяється безперервний ділянку пам'яті.

Змінні структурного типу можна створювати динамічно. Для цього потрібно описати покажчик на структуру

```
student * p;
```

потім створити екземпляр структури за допомогою операції new:

```
p = new student; //.
```

Доступ до елементів структури здійснюється за допомогою операції "точка", наприклад, щоб вивести на екран поле kurs структури stud2, можна використовувати оператор

```
cout << stud2.kurs; //.
```

Якщо звернення до структури відбувається через вказівник, то операцію "точка" використовують після розмінування вказівника (*p).kurs. Для цієї конструкції передбачений більш наочний аналог p->kurs операція "стрілка" звернення до поля kurs структури, на яку вказує покажчик p.

Таким чином, структури представляють собою спосіб організації даних в програмі. Обробка цих даних (якщо необхідно) відбувається за допомогою функцій. Подальшим розвитком способів програмування стало об'єднання даних з обробними їх під програмами і поява понять "клас" і "об'єкт".

Класи і об'єкти

Клас це такий "тип даних", який являє собою об'єднання полів (аналогічно структурі) і підпрограм (функцій). Підпрограми, що належать класу називаються методами. Загальний вигляд опису класу схожий на опис структури

```
1. class ІмяКласа {
2. Тип1 імя1;
3. Тип2 імя2;
4. ...
5. Функція1
6. Функція2
7. ...
8. }; //.
```

Доступністю включених в клас елементів можна управляти за допомогою службових слів private, public, protected. Словом public позначаються елементи класу, доступні для інших об'єктів програми, словом private елементи, доступні тільки для методів цього класу, protected елементи, доступні всередині методів класу і всіх його

спадкоємців. Як приклад опишемо клас для роботи з комплексними числами. Комплексне число складається з двох компонент дійсної і уявної частин. Значить клас повинен як мінімум містити два поля типу `float`

```
1. class CComplex {
2. public:
3. float x;
4. float y;
5. }; //.
```

Однак, таке визначення нічим не відрізняється від структури. Розширимо клас двома методами для обчислення модуля і аргументу комплексного числа

```
1. class CComplex {
2. public:
3. float x;
4. float y;
5. float Modul (void)
6. {Return sqrt (x * x + y * y);}
7. float Argum (void)
8. {If (x == 0) return 3.14159 / 2; else
   return atan (y / x);}
9. }; //.
```

Тут реалізація обох методів приведена всередині опису класу. Такий спосіб є зручним тільки для невеликих класів. Якщо методів досить багато, доцільно відокремити опис класу від його реалізації. Тоді в описі вказується тільки прототип методу, наприклад,

```
1. class CComplex {
2. public:
3. float x;
4. float y;
5. float Modul (void);
6. float Argum (void);
7. };
```

воно розміщується у заголовку `complex.h`. Реалізацію методів можна розмістити в окремому файлі `complex.cpp` і оформити наступним чином:

```

1. #include "complex.h"
2. float CComplex :: Modul (void)
3.     {Return sqrt (x * x + y * y);}
4. float CComplex :: Argum (void) {if
5.     (x == 0) return 3.14159 / 2;
6.     else return atan (y / x);} //.
```

Також як при оголошенні структури, визначення класу не призводить до виділення пам'яті. Воно є описом нового типу, який разом з даними володіє підпрограмами їх обробки. Змінні цього типу називаються об'єктами. Створимо об'єкт з ім'ям А, що представляє собою комплексне число CComplex А; //.

Об'єкт це змінна типу клас. Така термінологія використовується не випадково. Об'єктно-орієнтована технологія програмування дозволяє будувати моделі реальних об'єктів, а також вибудовувати їх ієрархії за допомогою наслідування властивостей одних об'єктів їх спадкоємцями. У зв'язку з цим корисно згадати відому з біології класифікацію тварин. Так, наприклад, клас ссавців має деякі властивості, властивими абсолютна всім його представникам. Разом з тим серед ссавців виділяють загін хижих. Вони також мають свої властивості. Серед хижих є сімейство ведмежих зі своїми властивостями. Таким чином, "конкретний екземпляр" ведмідь бурий має властивості всіх перерахованих рівнів класифікації. Іншими словами, кожен наступний рівень класифікації додає свої властивості. Ведмідь бурий є аналогом об'єкта, а рівні класифікації це класи зі структурою успадкування.

Як приклад побудуємо клас, що описує точку на екрані монітора. Щоб охарактеризувати точку необхідно вказати її координати два цілих числа а і b. Точка може переміщатися стрибком в нові координати x, y. Тому вона повинна володіти методом переміщення move, що змінює її координати. Опишемо клас точки:

```

1. class TPoint {
2. public:
3.     int a;
4.     int b;
5.     void move (int x, y);
6. };
```

```

7. TPoint :: move (int x, y)
8. {A = x; b = y;} //.

```

Далі побудуємо ще один клас, що описує видиму точку. Крім координат і можливості переміщення точка повинна мати характеристику видимості `int visible`. Ес-ли `visible = 0`, то точка невидима, в іншому випадку видима. Для управління видимістю передбачимо два методи `show` і `hide`.

```

1. class TVisiblePoint: public TPoint {public:
2. int visible;
3. void show (void);
4. void hide (void);
5. };
6. TVisiblePoint :: show ()
7.     {Visible = 1;}
8. TVisiblePoint :: show ()
9.     {Visible = 0;} //.
10.

```

Перший рядок опису класу задає спадкування класом `TVisiblePoint` всіх елементів класу `TPoint`. Слово `public` при описі спадкування означає, що похідному класу `TVisiblePoint` будуть доступні всі елементи базового класу `TPoint`.

І, нарешті, побудуємо ще один клас, що описує кольорову видиму точку. Очевидно, цей клас буде спадкоємцем класу `TVisiblePoint`, тільки матиме своє поле, що характеризує колір точки. Для простоти будемо вважати, що колір задається номером цілого типу. Тоді приходимо до наступного опису класу

```

1. class TColorVisiblePoint: public
   TVisiblePoint {public:
2. int color;
3. void setcolor (int c);
4. };
5.
6. TColorVisiblePoint :: setcolor (int c)
7.     {Color = c;} //.

```

Тут метод `setcolor ()` служить для перефарбування точки в інший колір. Тепер можна створити об'єкт точки.

```
TColorVisiblePoint ExamplePoint;
```

і використовувати її:

1. `ExamplePoint.x = 10;` // координата x
2. `ExamplePoint.y = 10;` // координата y
3. `ExamplePoint.show ();` // показати
4. `ExamplePoint.move (50,50);` // перемістити
5. `ExamplePoint.setcolor (5);` //
перекрасити.

На перший погляд здається, що використання об'єктно-орієнтованого програмування призводить до зайвого нагромадження коду. Однак, таке враження справедливо при програмуванні досить маленьких навчальних прикладів. При створенні об'ємних програм, а особливо при роботі колективу програмістів застосування об'єктів може дати суттєву перевагу в продуктивності праці програміста. Взагалі вважається, що недоцільно абсолютно всі програми робити об'єктними. Об'єктну технологію потрібно застосовувати там, де це зручно.

Конструктори і деструктори

При створенні об'єкта часто необхідно задати початкові значення його полів, виконати початкову ініціалізацію об'єкта. Для цього служить спеціальний метод, званий конструктором. Його ім'я завжди збігається з ім'ям класу. Конструктор може залежати від параметрів, які можна використовувати як початкові значення полів. Напишемо конструктор для класу `TColorVisiblePoint`

```
TColorVisiblePoint :: TColorVisiblePoint (int X, Y, C) {x = X; y = Y; color = C;} //
```

Тут задані початкові значення координат і кольор точки. Тепер при створенні об'єкта необхідно вказувати ці початкові значення, наприклад так:

// створюємо

```
TColorVisiblePoint * ExamplePoint = new TColorVisiblePoint (10,10,5);
```

```
ExamplePoint-> show (); // показуємо
```

```
ExamplePoint-> move (50,50); // переміщуємо.
```

Деструкція виконує функцію, зворотну конструктору. При видаленні об'єкта викликається метод-деструктор, ім'я якого збігається з ім'ям класу, але спочатку приписана тильда `~`. В основному деструктори застосовують в тому випадку, коли

виконувалося динамічне створення внутрішніх об'єктів або динамічне виділення пам'яті. Тоді всередині деструктора необхідно звільнити зарезервовану пам'ять.

Інкапсуляція, успадкування, поліморфізм

Об'єктно-орієнтовані мови програмування забезпечують механізми, які допомагають реалізовувати об'єктно-орієнтовану модель. До них відносяться інкапсуляція, успадкування і поліморфізм.

Інкапсуляція це механізм захисту даних від несанкціонованого зміни. При описі класу програміст виділяє частини класу, доступні зовнішній програмі, іншим об'єктам (public) і недоступні їм (публічному чаті protected). Доцільно якомога більше полів робити закритими, а для взаємодії з ними передбачати відкриті методи. При такій організації класу кажуть, що закриті частини вміщені в класі. Вважається, що інкапсуляція призводить до збільшення надійності програм, забороняючи несанкціонованих змін даних. Крім того, інкапсуляція дозволяє виділити інтерфейсну частину класу, призначену для взаємодії об'єкта з навколишнім середовищем і реалізацію класу. При незмінній інтерфейсній частини реалізація класу може змінюватися від версії до версії, не впливаючи на працездатність інших компонентів програми.

Спадкування це механізм такої організації відносин між класами, коли один клас може використовувати поля і методи іншого класу, а в результаті один об'єкт набуває властивостей іншого. Батьківський клас називають базовим, дочірній похідним.

Поліморфізм це механізм використання однакових імен для методів, об'єктів. Існує кілька проявів поліморфізму в об'єктно-орієнтованому програмуванні. Перше з них пов'язане із застосуванням однакових імен методів в батьківському і дочірньому класах. У цьому випадку говорять, що дочірній метод перекриває батьківський. Який з методів буде викликаний залежить від контексту його використання. Другий прояв також пов'язано із застосуванням однакових імен методів, але мають різні типи параметрів і результатів. Це дозволяє знайти третє застосування поліморфізму – зберігання даних різних типів в полях об'єкта з однаковими іменами.