МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

ПРОГРАМУВАННЯ
СТРУКТУРНИЙ ПІДХІД

МЕТОДИЧНІ ВКАЗІВКИ ДО КОМП'ЮТЕРНОГО ПРАКТИКУМУ

ДЛЯ СТУДЕНТІВ ФІЗИКО-ТЕХНІЧНОГО ІНСТИТУТУ
НТУУ "КПІ"

Київ
НТУУ «КПІ»
2011

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

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

ПРОГРАМУВАННЯ
СТРУКТУРНИЙ ПІДХІД

МЕТОДИЧНІ ВКАЗІВКИ ДО КОМП'ЮТЕРНОГО ПРАКТИКУМУ

**ДЛЯ СТУДЕНТІВ ФІЗИКО-ТЕХНІЧНОГО ІНСТИТУТУ
НТУУ "КПІ"**

Затверджено Методичною радою ФТІ НТУУ «КПІ»

Київ
НТУУ «КПІ»
2011

Програмування. Структурний підхід. Методичні вказівки до комп'ютерного практикуму. Для студентів І курсу Фізико-технічного інституту НТУУ "КПІ". / Уклад.: Н.М. Куссуль, А.Ю. Шелестов, А.М. Лавренюк, С.В. Скакун, О.М. Куссуль, А.В. Колотій. — К.: НТУУ "КПІ", 2011. — 120 с.

Гриф надано Методичною радою ФТІ НТУУ «КПІ»
(Протокол № ___ від _____ р.)

ПРОГРАМУВАННЯ. СТРУКТУРНИЙ ПІДХІД. МЕТОДИЧНІ ВКАЗІВКИ ДО КОМП'ЮТЕРНОГО ПРАКТИКУМУ

ДЛЯ СТУДЕНТІВ І КУРСУ ФІЗИКО-ТЕХНІЧНОГО ІНСТИТУТУ НТУУ "КПІ"

Укладачі: Н. М. Куссуль
А. Ю. Шелестов
А. М. Лавренюк
С. В. Скакун
О. М. Куссуль
А. В. Колотій

Відповідальна за випуск: Т. В. Литвинова, канд. техн. наук, доц.

Рецензент: О. М. Кравченко, канд. техн. наук, н.с.

За редакцією укладача

03056, м. Київ-56, просп. Перемоги, 37

Зміст

Комп'ютерний практикум №1
Принципи створення програм на мові С++

Мета роботи: отримати навички створення програм на мові С++.

1.1. Теоретичні відомості

Структура програми

Програма на мові С++ є блочно-структурованою та, як правило, містить деякий набір функцій. *Функція* — це іменована частина програми, до якої можна звертатися з її інших частин, вказуючи її ім'я, наприклад, `SimpleFunc(5,1.9)`. У даному випадку функції з іменем `SimpleFunc` передається два числових параметра. Питання створення та використання функцій більш докладно розглядаються у лабораторному практикумі № 3.

До складу кожної програми на мові С++ повинна входити *головна функція* `main()`. Саме ця функція є початковою точкою входу в програму.

Крім функції `main()` до програми може входити будь-яка кількість функцій. Кожна функція по відношенню до іншої є зовнішньою, тобто жодна з функцій не може міститися всередині іншої. Для того, щоб функція була доступна, необхідно, щоб до її виклику про неї було відомо компілятору.

Основну структуру програми на мові С++ наведено на рис. 1.1. Нижче наведено приклади двох простих програм на мові С++ (приклади 1.1 та 1.2).

Приклад 1.1. Програма виводу на екран привітання "Hello, world!"

```
#include <iostream>
// директива препроцесора
using namespace std;
// включає в програму визначення
// стандартного простору імен
int main()
// заголовок функції main()
```

```
{
  // початок тіла функції main()
  cout<<"Hello, world! \n”;
  // \n - символ нового рядка
  cin.get();
  //очікується натискання будь-якої клавіші
  return 0;
  //оператор повернення завершує функцію main()
}//кінець тіла функції main()
```

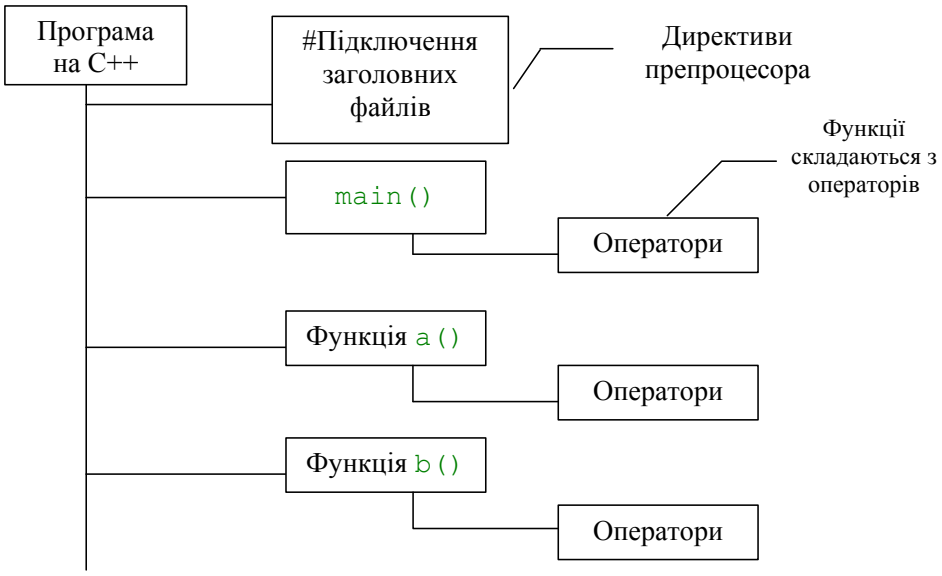


Рис.1.1 Структура програми на С++

Приклад 1.2. Програма вводу з клавіатури та виводу на екран цілого числа

```
#include <iostream>
// директива препроцесора
using namespace std;
// включає в програму визначення
// стандартного простору імен
int main()
{
  int a;
  // оголошення змінної a цілого типу
  cin>>a;
```

```
    //зчитування змінної a з стандартного потоку
    //введення
    cout<<"Ви ввели число "<<a<<endl;
    //запис тексту та змінної a в стандартний поток
    //виведення
    return 0;
    //оператор повернення завершує функцію main()
}
```

У наведеному прикладі `endl`— це маніпулятор виведення, що вставляє у вихідний потік символ переходу на новий рядок (аналогічно `\n`).

Засоби введення/виведення

При запуску програми на мові С++ автоматично створюється декілька стандартних потоків, зокрема: `cin` (стандартний потік вводу з клавіатури) та `cout` (стандартний потік виводу на екран).

Існує ще 2 стандартних потоки `cerr` та `clog`, що призначені для виводу помилок. Для того щоб використовувати ці потоки, достатньо підключити заголовний файл `<iostream>` та вказати стандартний простір імен, як це показано у прикладі 1.3.

Приклад 1.3

```
#include <iostream>
using namespace std;

int main()
{
  int i;
  cout<<"Input i : ";
  cin>>i;
  cout<<"i = " <<i;
  ...
}
```

У наведеному прикладі вводиться число, а після цього воно виводиться на екран.

Потоки є засобом вводу/виводу мови програмування С++. Більш докладно можливості їх використання будуть розглядатися у лабораторному практикумі № 5 та у курсі **Програмування. Об'єктно-орієнтований**

підхід. Крім потоків для вводу/виводу інформації можна використовувати також функції мови С, зокрема функцію `printf()` та `scanf()`. Більш докладно ці засоби також будуть розглядатися у лабораторному практикумі № 5.

Етапи створення виконуваного коду програми

Для того, щоб забезпечити перетворення програмного коду на мові С++ у виконуваний файл, потрібно виконати наступні дії:

1. Скориставшись текстовим редактором, написати програму і зберегти її у файлі на диску. Цей файл містить код програми.

2. Скомпілювати код програми.

Цей процес є двоетапним. Зокрема, при компіляції програми на мові С++ спочатку виконується препроцесінг, а після цього — компіляція в об'єктний код.

3. Зв'язати об'єктний код основної програми (отриманий в процесі компіляції) з додатковим об'єктним кодом бібліотечних та/або користувацьких функцій і, таким чином, скомпонувати єдину програму. Програми С++ зазвичай використовують бібліотечні функції, для яких вже існує об'єктний код, наприклад, `sin()`, `cos()`, `sqrt()`, тощо. Під час компоновки об'єктний код програми об'єднується з об'єктним кодом бібліотечних функцій, які використовуються в програмі, та стандартним кодом початкового завантаження. (Код початкового завантаження забезпечує можливість виклику функції `main()` із середовища операційної системи.) В результаті створюється виконуваний файл.

Перед основною компіляцією виконується попередня обробка (препроцесінг) програмного коду. При цьому всі директиви препроцесора, що починаються з символу `#`, обробляються однаково, а саме замість самої директиви в програмний код вставляється контент відповідного заголовного файлу. Заголовні файли зазвичай містять необхідні описи бібліотечних функцій, змінних та структур даних. Код, отриманий після виконання препроцесінгу, називається одиницею трансляції.

Препроцесор — це програма, яка обробляє вихідний код перед основною компіляцією. Препроцесор обробляє директиви, що починаються з символу `#`, та запускається автоматично перед компіляцією програми.

```
#include <file1>
 #include <file2>
```

```
...
#include <filen>
```

Слід звернути увагу на те, що на відміну від більшості конструкцій мови С++, ця директива не завершується крапкою з комою. Використання таких директив призводить до того, що препроцесор підставляє на місце цих директив тексти файлів у відповідності з тими, що перелічені у дужках `<...>`. Якщо ім'я файлу міститься у таких дужках, то пошук файлу буде проводитися у спеціальному каталозі заголовних файлів (як правило, це каталог `INCLUDE`).

Для підключення власних заголовних файлів імена цих файлів вказуються у подвійних лапках, наприклад, `"file.h"`. Тоді пошук файлу ведеться у поточному каталозі активного диску. Якщо пошук не дозволив знайти потрібний файл, пошук продовжується у стандартному каталозі для заголовних файлів.

Деякі реалізації мови С++, наприклад, Microsoft Visual C++, Borland C++, реалізовані у вигляді інтегрованих середовищ розробки, що дозволяють виконувати всі вищеописані етапи створення виконуваного файлу в автоматичному режимі. Інші реалізації, такі як AT&T C++ або GNU C++ для операційних систем Unix та Linux, дозволяють виконувати тільки етапи компіляції та компоновки, а команди в цих реалізаціях вводяться в командному рядку системи. В таких випадках для створення і модифікації програмного коду можна використовувати будь-який доступний текстовий редактор (наприклад, в Unix можна використовувати редактори `vi, ed`, `emacs`).

При необхідності компілятор можна запускати в командному рядку операційної системи Windows. Наприклад, компілятор Borland Builder C++ можна запустити в командному рядку наступним чином:

```
bcc32  unit1.cpp
```

а компілятор Visual Studio:

```
mc unit1.cpp
```

У будь-якому випадку, для отримання довідкової інформації щодо параметрів командного рядка компілятора можна використати параметр `/?` або `-h`.

Якщо компіляція пройшла успішно, буде створено виконуваний файл, наприклад `unit1.exe`.

Операції і вирази в С++

Вираз — це послідовність символів операцій, операндів і символів пунктуації, що задає обчислювальний процес знаходження результату певного типу. *Найпростішим виразом* є константа, змінна або виклик функції.

Операції і вирази мови С++ дозволяють задати певну послідовність дій. *Прості вирази* містять символ операції та операнди. Прикладом простого виразу з *бінарною* операцією є віднімання: `3.4 - x`.

Операнд є елементом-учасником операції. Операндами можуть бути *константи*, *змінні*, *виклики функцій* і *вирази*.

У мові С++ визначено наступні основні арифметичні операції.

Символи `+`, `-`, `*`, `/` — символи *бінарних* операцій додавання, віднімання, множення і ділення відповідно. Існує два види операції ділення: *цілочисельне ділення* (ділення з остачею) і ділення *без остачі* (результат такої операції є дійсним числом). Тип операції ділення, що буде використано, визначається типом операндів.

Для цілих чисел визначена операція `%` — ділення *по модулю* (знаходження остачі від ділення). Наприклад, `5%2 = 1`.

Прикладом унарної *арифметичної* операції є *операція зміни знака числа*.

Компілятор по *імені операції* і *типам операндів* визначає можливість виконання операції та операції, що повинні бути виконані. Виконання *операції дозволяє* отримати результат певного типу, що залежить від типів задіяних операндів.

Виклик функції— це використання імені функції, за яким у круглих дужках міститься список аргументів (може бути порожній). Після виклику функції вона буде виконана з відповідними аргументами. Прикладом виразу з викликом функції може бути наступний:

```
a*sin(x+k)+b*cos(x-k)
pow(2, n+1)-1
```

Символ `=` означає бінарну операцію *простого присвоювання*, у результаті виконання якої значення правого операнду присвоюється лівому операнду.

Порядок обчислення виразу визначається розташуванням знаків операцій, круглих дужок і *пріоритетами* виконання операцій. Вирази із найвищим пріоритетом обчислюються першими.

Якщо у виразі міститься декілька операцій одного пріоритету на тому самому рівні, то їх обробка виконується відповідно до порядку виконання: зліва направо чи справа наліво.

Оператори закінчуються крапкою з комою. Прості оператори можна об'єднати в складений *блок* за допомогою фігурних дужок, за якими крапку з комою можна не ставити.

Типи даних

Під *типом даних* розуміють множину допустимих значень цих даних і множину дозволених операцій над ними. Водночас тип даних визначає і розмір пам'яті, що займають змінні і константи даного типу. Пам'ять виділяється не для типу даних, а виділяється для розміщення змінної або константи заданого типу.

У мові С++ виділяють ***вбудовані та користувацькі типи даних***.

Вбудовані типи - це типи, які підтримує мова програмування.

Вбудовані типи бувають простими (базовими) та похідними, що утворюються від базових.

До простих типів відносяться наступні:

- `bool`
- `int`
- `char`
- `float`
- `double`

До похідних типів відносяться наступні:

- масиви (`int a[5]`)
- вказівники (`int*a`)

Існує також порожній тип `void` (не має значення).

Прості типи мають набір значень і представлень, прив'язаних до архітектури машини, на якій працює транслятор.

У С++ *прості типи можуть бути модифіковані за допомогою ключових слів*:

- `short`
- `long`
- `signed`
- `unsigned`

Крім вбудованих типів у мові С++ існують користувацькі типи, які вимагають попереднього оголошення. Як правило, такі типи є складеними:

- `struct` (структура)
- `union` (об'єднання)
- `enum` (перерахування)
- `class` (класи)

Змінні і константи цілих типів також можуть оголошуватись за допомогою модифікаторів `signed` і `unsigned` (знакове, беззнакове). При використанні модифікаторів `short` і `long` дозволяється опускати ім'я типу `int`. Типи з плаваючою точкою або дійсні типи представлені трьома модифікаціями, що характеризують точність представлення дійсних чисел: `float` — одиничної точності; `double` — подвійної точності; `long double` — розширеної точності.

Для визначення розміру об'єкту програми в певних одиницях (для простих типів — в байтах), використовується оператор `sizeof`:

```
cout <<sizeof(int);
```

Діапазони всіх типів змінних (максимальні й мінімальні значення) можна одержати з заголовочного файлу `climits` (в більш ранніх реалізаціях - `limits.h`).

Оголошення змінних

Дані в програмі можна розділити на змінні і константи. Перед використанням змінні і константи повинні бути оголошені за допомогою оператора оголошення.

```
[<специфікатор класу пам'яті>] [const] <специфікатор типу> <ідентифікатор> [= <початкове значення>] ,
[<ідентифікатор> [= <початкове значення>]] ;
```

Наприклад,

```
int a=5, y;
const float g = 9.81,  C = 0.577216;
```

Ключове слово `const` вказує, що записані праворуч ідентифікатори є *константами* (константними змінними). При цьому значення константи задається обов'язково і у програмі змінюватися не може.

Зі змінною пов'язані поняття її оголошення, визначення та ініціалізації. ***Оголошення*** змінної дозволяє зв'язати тип з її ім'ям (наприклад, `int x`).

При ***визначенні*** змінної для неї виділяється пам'ять. Більшість оголошень є також визначеннями (наприклад, `int x`). ***Ініціалізація*** змінної полягає у присвоєнні їй початкового значення (наприклад, `int x=1`). Приклади оголошення, ініціалізації та визначення змінних можна побачити у наступному прикладі:

```
char  ch;
int  count =1;
const double  pi=3,141592;
extern int error_number;
/*оголошення, але не визначення, змінна визначена в
                    іншому файлі*/
char *name =”Njal”; // рядок (масив символів)
char *season[]={“spring”, “summer”, “autumn”,
“winter”}; // масив рядків
double  sqrt(double); // функція
```

Умовний оператор

Умовний оператор відноситься до категорії *операторів керування* та забезпечує виконання або невиконання деякого оператора або групи операторів в залежності від заданої умови. Оператор `if` є одним з самих популярних засобів, які змінюють звичайний порядок виконання операторів програми. Він використовується в одній з наступних форм:

```
if (умовний вираз) оператор1;

if (умовний вираз) оператор1;
else оператор2;
```

Після ключового слова `else` також може стояти оператор `if`, таким чином утворюючи багаторівневий оператор керування:

```
if (умовний вираз) оператор1;
else if оператор2;
else if оператор3;
…
else if операторn;
else операторn+1;
```

Якщо значення умовного виразу істинне (не нуль), то виконується `оператор1`; якщо — хибне (рівне нулю), то `оператор1` пропускається, а

виконується `операор2` після ключового слова `else`. Іноді після перевірки умови необхідно виконати цілу групу операторів. Тоді ту частину програми, яку треба виконати після `if`, можна вказати у фігурних дужках `{}`. Нижче проілюстровано можливості використання умовного оператора.

Нижче за допомогою умовного оператора знаходиться мінімум з двох чисел *x* та *y*:

```
  if(x<y) min=x;
else min=y;
cout<<"min="<<min;
```

Перевірка коректності вводу змінної, яка має знаходитися у діапазоні від 1 до 31, може виглядати так:

```
  cin>>den;
if((den<1)||(den>31)) cout<<"Помилка вхідних дани";
```

Пошук максимуму з трьох чисел *a*, *b*, *c* можна реалізувати наступним чином:

```
if((a>b)&&(a>c)) max=a;
else if(b>c) max=b;
else max=c;
cout<<"max="<<max;
```

Додатковий теоретичний матеріал. Системи числення та перетворення із однієї системи числення в іншу

У двійковій системі числення користуються лише двома цифрами: 0 і 1. У вісімковій - цифрами від 0 до 7, а в шістнадцятковій - цифрами від 0 до 9 і літерами від 'A' до 'F', які відповідають десятковим числам від 10 до 15.

Для переведення з десяткової системи числення у двійкову (вісімкову, шістнадцяткову, …) користуються діленням у стовпчик. При діленні числа *n* на 2 під числом *n* записуємо остачу від ділення *n* на 2, під двійкою – частку. Процес ділення закінчується, коли частка стане рівною 1. Далі слід записати останню частку (одиницю) і всі остачі від ділення у зворотному порядку. Наприклад, знайдемо двійкове представлення числа 20:

20	2			
0	10	2		
	0	5	2	
		1	2	2
			0	1

Записавши остачі у зворотному порядку, отримаємо: $20_{10} = 10100_2$.

Знайдемо шістнадцяткове представлення числа 511:

511	16	
15=F	31	16
	15=F	1

З таблиці отримаємо: $511_{10} = 1FF_{16}$.

Якщо b – основа системи числення, то числу n, що має в ній запис $\overline{a_n a_{n-1} \ldots a_1 a_0}$, у десятковій системі відповідає число:

$$n = a_n * b^n + a^{n-1} * b^{n-1} + \ldots + a_1 * b + a_0 .$$

Приклади переведення чисел з різних систем числення в десяткову:
1111 із двійкової: $1111_2 = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 = 15_{10}$
16 з вісімкової: $16_8 = 1 * 8^1 + 6 = 14_{10}$
FF із шістнадцяткової: $FF_{16} = 15 * 16^1 + 15 = 255_{10}$

1.2. Порядок виконання роботи

1. Проаналізувати умову задачі.
2. Розробити алгоритм та створити програму розв'язання задачі згідно з номером варіанту.
3. Результати роботи оформити протоколом.

1.3. Варіанти завдань

Обов'язкові завдання

1. Якщо серед трьох чисел a, b, c є хоча б одне парне, знайти максимальне, інакше – мінімальне.
2. Ввести a>=1. Знайти значення якого з виразів більше: $1/a$ чи $\sin(a)$.
3. Ввести два числа. Менше замінити півсумою, а більше - подвоєним добутком.

4. Ввести три числа *a*, *b*, *c*. Подвоїти кожне з них, якщо *a*>=*b*>=*c*, інакше поміняти значення *a* та *b*.

5. Визначити чи є точка з координатами *x*, *y* точкою перетину діагоналей квадрата зі стороною *r*, одна вершина якого розташована в початку координат.

6. Визначити значення функції в залежності від значення аргументу:

$$y=\begin{cases}\dfrac{1}{x}, & якщо\ \ x>10\\ ax^2, & якщо\ \ -10\le x\le 10\\ \sin(x), & якщо\ \ x<-10\end{cases}$$

Обчислити значення наступних виразів. При цьому знайти область визначення функцій та забезпечити необхідну реакцію програми на некоректні ситуації.

7) $y=\dfrac{\sqrt{x-4}}{\sqrt{3a^3}}$;

8) $y=\log_4(x^2-4)+\dfrac{1}{x}$;

9) $y=\sqrt{\ln\left(\dfrac{1-a}{x}\right)}$;

10) $y=\dfrac{\sin(2x^3)}{x+2}+\cos\dfrac{x}{2a}$;

11) $y=\dfrac{\sqrt{x^2-4}}{2a\cos x}$;

12) $y=\dfrac{\sqrt{x+3}+x^3+3}{(x-1)^2}+\dfrac{\cos x^3}{2}$;

13) $y=\cos x^3+\dfrac{\sqrt{x}}{\sin a}$;

14) $y=tg\dfrac{1}{x-a}-\sqrt{2x+3a}$;

15) $y=x^3+\dfrac{1}{\sqrt{a}}$;

16) $y=\dfrac{|x-b|}{2\sqrt{a}}-\dfrac{tgx}{b^2}$;

17) $y=\dfrac{\sqrt{x^2+4a}-\sqrt[3]{x}}{1-\dfrac{x^2}{2}}$;

18) $y=\dfrac{x^3+x+3}{a-2}+\sqrt{\dfrac{(a-x)^5}{(2x-a)^3}}$.

1.4. Завдання до додаткового теоретичного матеріалу

1. Знайти двійкове, вісімкове та шістнадцяткове представлення десяткового числа 31.
2. Знайти шістнадцяткове представлення десяткового числа 1000.
3. Знайти десяткове представлення двійкого числа 11001.
4. Знайти десяткове представлення двійкого числа 01000.

1.5. Контрольні запитання

1. Яка структуру має програма на мові С++?
2. В чому полягають кроки перетворення програмного коду на мові С++ у виконуваний файл?
3. Що таке препроцесор? Як отримати одиницю трансляції?
4. Пояснити зміст поняття оператор.
5. Що розуміється під типом даних?
6. Які типи даних використовуються у мові С++?
7. Яким чином розрізняються змінні за часом життя?
8. Дайте означення виразу.
9. Приведіть приклади операцій з однаковим пріоритетом.
10. Вкажіть операції з найвищим та найнижчим пріоритетом.
11. Які засоби мови С++ призначені для вводу/виводу даних?

Комп'ютерний практикум №2
Умовна операція, множинний вибір та оператори циклів

Мета роботи: отримати навички роботи з умовними операторами, операторами множинного вибору та циклів.

2.1. Теоретичні відомості

Операція умови ?

В мові С++ є короткий засіб запису оператора `if... else`. Для цього використовують тернарну умовну операцію. Вона має наступну форму запису:

```
(умовний вираз) ? вираз1 : вираз2
```

Якщо умовний вираз істинний, то виконується `вираз1`, якщо хибний — `вираз2`. Умовну операцію зручно використовувати у випадках вибору значення з двох можливих. Наприклад, знайти максимум з двох чисел *x* і *y* із використанням операції `?` можна наступним чином:

```
max=(x>y) ? x:y;
cout<<"max="<<max;
```

Множинний вибір: оператори `switch` і `break`

Іноді виникає необхідність вибору одного варіанту з декількох. Зручним засобом вибору з множини варіантів є оператор `switch`, який має наступну форму запису:

```
switch (вираз)
{
  case константа1: оператор1; break;
  case константаN: операторN; break;
  default : оператор;
}
```

Вираз, який стоїть у дужках після `switch` має бути цілочисельним.

Оператор вибору працює наступним чином. Спочатку підраховується вираз, який стоїть у дужках після `switch`. Далі виконується перехід на одну з міток, позначену ключовим словом `case`, значення константи після якої дорівнює виразу в дужках після `switch`. Константа, що стоїть після `case`, повинна бути цілого типу. Якщо вираз в дужках не дорівнює жодній з констант, які перевіряються, то виконується перехід на мітку `default` (не є обов'язковою).

Зазвичай дія кожної гілки закінчується оператором `break`. Виконання цього оператора призводить до виходу з оператора `switch`. Якщо `break` відсутній, то керування передається наступному оператору, позначеному `case` або `default` і так далі, поки не зустрінеться оператор `break`.

Ключові слова `case` і `default` не можуть знаходитися за межами блоку `switch`.

Оператори циклів

При виконанні програми часто виникає необхідність неодноразового повторення однотипних обчислень над різними даними. Для цих цілей використовуються оператори циклів.

Цикл представляє собою частину програми, у якій одні й ті самі обчислення виконуються неодноразово над різними значеннями одних й тих самих змінних (об'єктів).

Для організації циклів в C++ використовуються наступні три оператора: `while`, `for` і `do – while`.

Цикл `while`

Цикл `while` є циклом з передумовою. Він використовується у випадку, коли, по-перше, не відома точна кількість повторів і, по-друге, при цьому немає необхідності, щоб цикл неодмінно був виконаний хоча б один раз. Цикл `while` має наступну синтаксис:

```
while (вираз)
   оператор;
```

В якості виразу зазвичай використовуються умовні вирази. В загальному випадку можна використовувати вирази довільного типу. На місці оператора може стояти простий оператор або сукупність операторів, об'єднаних у блок дужками `{}`.

Якщо вираз істинний (не рівний нулю), то тіло циклу виконується один раз, далі вираз перевіряється знову. Ітерації (перевірка умови та тіло циклу) виконуються до тих пір, поки вираз не стане хибним (рівним нулю).

При організації циклу `while` в його тіло повинні бути включені конструкції, які б змінювали вираз, що перевіряється, так, щоб все ж таки він став хибним. В протилежному випадку виконання циклу ніколи не закінчиться.

У фрагменті нижче з використанням циклу `while` користувачу надається 10 спроб на те, щоб вгадати число.

```
  int i=1, rez=1;
while (i++<=10&&rez!=25)
{
  cout<<"\nВведіть число:" ;
  cin>>rez;
}
if (i==12&&rez!=25)cout<<"\nВи не вгадали.";
else cout<<"\nВітаю! Ви вгадали число.";
```

В даному прикладі цикл виконується до тих пір, поки не вгадано число або не вичерпано кількість спроб.

Цикл `do–while`

Цикл `do–while` є циклом з постумовою і використовується у тих випадках, коли невідома точна кількість повторів, але водночас цикл необхідно виконати не менше одного разу. Цикл `do–while` дуже схожий на цикл `while`; різниця тільки в тому, що перевірка істинності виразу в циклі `do–while` виконується після виконання тіла циклу. Цей цикл має наступну форму запису:

```
do оператор; while (вираз);
```

Керуючі оператори в циклах

Існують ще три оператора, призначених для керування порядком виконання програми на мові C++.

Оператор `break` є найбільш важливим з цих трьох операторів. Оператор `break` може використовуватися в циклах всіх трьох типів. Виконання оператора `break` призводить до виходу з циклу, в якому він знаходиться, і переходу до наступного за блоком циклу оператора. Якщо оператор `break`

знаходиться всередині вкладених циклів, то його дія поширюється тільки на той цикл, в якому він безпосередньо знаходиться.

Як приклад застосування даного оператору розглянемо можливе розв'язання задачі вгадування числа з 10 спроб.

```
i=1;
while( i++<=10 )
{
   cin>>rez;
   if (rez==15) break;
   cout<<"\nПощастить наступного разу.";
}
if ( i!=12 )  cout<<"\nВи вгадали!.";
```

В цьому прикладі завершення виконання циклу відбувається за допомогою оператора `break`.

Оператор `continue` може використовуватися тільки серед операторів тіла циклу. Цей оператор призводить до переходу до наступної ітерації без завершення поточної.

Розглянемо використання оператора `continue`. Вводяться числа місяця для обробки. Необхідно здійснити перевірку коректності вводу, коли число 31 буде кінцевим. Цю задачу можна розв'язати так:

```
while(den!=31)
{
   cin>>den;
   if (den<1||den>31) continue;
   … // Обробка числа den
}
```

В даному прикладі неправильне введення значення призводить до пропуску частини ітерації, призначеної для обробки цього значення.

Оператор `goto` (перехід на задану мітку) краще не використовувати, оскільки він призводить до значних ускладнень логіки програми.

Існує лише один випадок, коли програмісти-професіонали допускають використання `goto`, — це вихід з вкладеного набору циклів при виявленні помилок (`break` дає можливість виходу лише з одного циклу).

Цикл типу `for`

Цикл типу `for` є циклом з параметрами і зазвичай використовується у випадку, коли відома точна кількість повторів обчислень. При цьому виконуються три операції: *ініціалізація лічильників циклів*, *порівняння його*

значення з деяким граничним значенням і *зміна значення лічильника при кожному проходженні тіла циклу*.

Цикл `for` має наступну форму запису:

```
for (вираз1; вираз2; вираз3) оператор;
```

`Вираз1` обчислюється першим. Зазвичай тут виконується ініціалізація лічильників циклів і змінних. Цей вираз обчислюється один раз, коли цикл `for` починає виконуватися. Далі обчислюється `вираз2`. Він служить для перевірки умови. Якщо значення `виразу2` не є нулем, то виконується `оператор` (тіло циклу). Якщо ж значення `виразу2` рівне нулю, то цикл завершується. `Вираз3` обчислюється в кінці кожного виконання тіла циклу.

В якості оператора може використовуватися простий оператор або сукупність операторів, об'єднаних у блок дужками `{}`.

Підрахуємо y^T. Можливий варіант розв'язання цієї задачі має вигляд:

```
int i, rez=1;
for ( i=1; i<=T; i++ ) rez=rez*y;
cout<<"rez="<<rez;
```

Вкладені цикли

Вкладеним циклом називають конструкцію, в якій один цикл виконується всередині другого. Внутрішній цикл виконується повністю під час кожної ітерації зовнішнього циклу. Наприклад, треба заповнити весь екран символами `'#'`. Це можні зробити наступним чином.

```
  for (int i=1; i<=25; i++ )
for (int k=1;k<=80; k++ )
     cout<<'#';
```

В цій програмі 25 разів виконується виведення 80 символів.

В програмі можна використовувати будь-які комбінації вкладених циклів всіх типів: `while`, `for`, `do – while`, якщо цього потребує логіка побудови програми.

Розглянемо ще один приклад вкладеного циклу. Необхідно ввести десять значень днів місяця з перевіркою правильності вводу. Це можна зробити так:

```
for (int i=1; i<=10; i++ )
{
do cin>>den;
while(den<1||den>31);
```

```
cout<<den;
}
```

В даному прикладі зовнішній цикл виконується 10 разів, а внутрішній виконується до тих пір, поки не буде введено правильне значення.

Який оператор циклу кращий

Циклом якого типу краще всього користуватися? Спочатку слід вирішити, чи потрібен цикл з передумовою, чи з постумовою.

Існує ряд причин, за яких професійні програмісти віддають перевагу використанню циклів с передумовою. Перша — це те, що краще спочатку прийняти рішення, чи потрібно щось робити, а не після того, як це вже зроблено. Друга — те, що програма більш зрозуміла, коли умова, яка перевіряється, знаходиться спочатку, а не в кінці блоку циклу. Третя причина полягає в тому, що в більшості випадків необхідно, щоб тіло циклу ігнорувалося повністю, якщо умови не виконуються.

У випадку вибору циклу з передумовою виникає питання: що краще, `for` чи `while`? В принципі все, що можна зробити за допомогою одного циклу, можна зробити й за допомогою іншого. Виходячи з міркувань правильного стилю програмування використанню циклу `for` віддається перевага, коли в циклі використовується ініціалізація і корекція змінної. А цикл `while` зручніше використовувати, коли цього робити не треба.

В мові C++ оператор циклу `for` є більш гнучким засобом, ніж аналогічні оператори циклів в інших мовах програмування. Крім описаних вище, існує ще багато інших можливостей використання цього типу циклів. Приклади деяких з них наведено нижче.

Для обчислення y^5 можна скористатися циклом `for` наступного вигляду.

```
…
int i,r;
for ( i=5, r=1; i>=1; i-- ) r=r*y;
cout<<"r="<<r;
…
```

Нижче наведено приклад циклу `for` з шагом змінної, не рівним 1.

```
…
for (int n=5; n<61; n+=15) cout<<n;
…
```

В якості лічильника циклу `for` можна використовувати і символи, наприклад наступним чином.

```
…
for (char chr='A'; chr<='Z'; chr++)  cout<<chr;
…
```

2.2. Приклади

Приклад 2.1. Використання оператора switch

Проаналізуємо значення змінної *rez*, яка є отриманою оцінко, та трансформуємо її в інше представлення.

```
...
switch (rez)
{
   case 5: cout<<"Оцінка – відмінно."; break;
   case 4: cout<<"Оцінка – добре."; break;
   case 3: cout<<"Оцінка – задовільно."; break;
   case 2: cout<<"Оцінка – незадовільно."; break;
   default: cout<<"Невірне значення rez.";
}
...
```

Приклад 2.2. Використання оператора умови ?

Порівняти два значення змінних і вивести на екран значення більшої змінної. При цьому значення більшої змінної присвоюється змінній *z*.

```
#include <iostream>
using namespace std;
int main()
{
  int x,y,z;
  z=(x>y) ? x:y;
  cout << "z:" << z;
  cout << "\n"; //додавання нового рядка
  return 0;
}
```

Приклад 2.3. Програма для підрахунку середньої оцінки

```
#include <iostream>
using namespace std;
int main()
```

```
{
  int total=0,
      gradeCounter,
      grade,
      average;
  gradeCounter=1;
  while (gradeCounter <= 10)
  {
    cout << "Введіть оцінку:";
    cin >> grade;
    total=total+grade;
    gradeCounter=gradeCounter+1;
  }
  average=total/10;
  cout<<"Середня оцінка дорівнює"<<average<<endl;
  return 0;
}
```

Приклад 2.4. Використання циклу `for`

За допомогою циклу `for` вивести 5 ступенів введеного значення.

```
#include<iostream>
using namespace std;
int main()
{
  int a, b;
  cout<<"Input a: ";
  cin>>a;
  b=a;
  for(int i=0;i<5;i++)
  {
     cout<<b<<' ';
     b*=a;
  }
  return 0;
}
```

2.3. Порядок виконання роботи

1. Проаналізувати умову задачі.

2. Розробити алгоритм та створити програму розв'язання задачі згідно з номером варіанту, обравши задачі з частин 1, 2 та 3.
3. Результати роботи оформити протоколом.

2.4. Варіанти завдань

Обов'язкові завдання

Частина 1

Розв'язати наступні задачі, використовуючи умовну операцію:

1. Створити програму обчислення знаку числа, що вводиться з клавіатури.
2. Написати програму обчислення мінімуму із двох чисел.
3. Написати програму обчислення мінімуму із трьох чисел.
4. Написати програму обчислення абсолютного значення введеного числа.
5. Написати програму, яка потроює введене додатне число та підносить до квадрату від'ємне.

Частина 2

Розв'язати наступні задачі, використовуючи оператор множинного вибору.

1. Створити текстове меню, в якому при виборі першого пункту обчислюється значення квадрату введеного числа, при виборі другого пункту – значення кубу і т.д.
2. Створити текстове меню, в якому при виборі першого пункту виводиться привітання, при виборі другого пункту – запрошення до роботи на комп'ютері, при виборі третього — пропонується завершити роботу.
3. Створити текстове меню, в якому при виборі першого пункту обчислюється косинус введеного числа, при виборі другого пункту – синус, при виборі третього — тангенс.

Частина 3

Розв'язати наступні задачі двома способами: спочатку з використанням оператора циклу `while`, а потім — `for`.

1. Написати програму введення додатних чисел.

2. Написати програму знаходження всіх чисел кратних введеному та таких, що не перевищують 300.
3. Вивести члени арифметичної прогресії, що не перевищують 100, з заданим початковим членом та кроком.
4. Вивести перших 10 членів арифметичної прогресії з заданим початковим членом та кроком.
5. Вивести члени геометричної прогресії, що не перевищують 100, з заданим початковим членом та кроком.
6. Вивести перших 10 членів геометричної прогресії з заданим початковим членом та кроком.
7. Написати програму визначення максимального числа в послідовності цілих додатних чисел.
8. В послідовності чисел знайти добуток чисел, кратних 3.
9. Дано натуральне число n. Знайти факторіал цього числа (факторіал числа n – це добуток всіх натуральних чисел від 1 до n, тобто $n!=1\times2\times3\times\ldots\times n$).
10. Вводяться по черзі дані про зріст студентів групи. Визначити середній зріст студентів.
11. Написати програму знаходження степеня числа a з натуральним показником n.
12. Вивести всі парні числа від n до m.
13. Написати програму знаходження суми значень функції $y=x^2$ на відрізку [1,5] з кроком 1.
14. Написати програму виведення всіх чисел від 1 до 1000, які закінчуються цифрою 3.

Завдання підвищеної складності

1. Знайти суму чисел Фібоначчі, які не перевищують 1000 (числа Фібоначчі f_n обчислюються за формулами $f_0=f_1=1$; $fn=f_{n-1}+f_{n-2}$ при $n=2,3,...$ Числова послідовність Фібоначчі 1, 1, 2, 3, 5, 8, 13, ...).
2. Знайти перше число Фібоначчі, яке більше за m ($m>1$) (числа Фібоначчі f_n обчислюються за формулами $f_0=f_1=1$; $fn=f_{n-1}+f_{n-2}$при $n=2,3,...$ Числова послідовність Фібоначчі 1, 1, 2, 3, 5, 8, 13, ...).
3. Написати програму знаходження всіх досконалих чисел на заданому інтервалі (натуральне число називається досконалим, якщо воно дорівнює сумі своїх дільників, за винятком самого себе, наприклад, 6=1+2+3) .
4. Визначити, чи є задане шестизначне число "щасливим" (сума перших трьох цифр має дорівнювати сумі останніх трьох цифр)?

5. Дано натуральне чотирицифрове число n. З'ясувати, чи є воно паліндромом (таким, що читається однаково зліва направо та справа наліво, наприклад 1221).

6. Написати програму, яка серед двоцифрових чисел вибирає числа, рівні сумі своїх цифр. Навіть якщо відповідних варіантів немає, перевірити всі двоцифрові числа.

7. Написати програму знаходження суми n перших членів послідовності 1/2+3/4+5/6+...

8. Вивести 5 простих чисел, які більші введеного числа.

9. Написати програму, що підраховує кількість цифр у введеному цілому числі n.

10. Знайти суму ряду, загальний член якого заданий за формулою $A_n=(x*n)/n!$.

11. Дано послідовність із n цілих чисел. Написати програму, яка обчислює добуток максимального та мінімального елементів цієї послідовності.

12. Написати програму, яка виводить всі дільники цілого числа у порядку зростання.

13. Написати програму, яка виводить всі трьохзначні числа, рівні сумі кубів своїх цифр.

14. Дано натуральне число n. Написати програму, яка виводить число, записане цифрами числа n в зворотному порядку.

15. Знайти мінімальне значення функції $y=\sin(x)*x$, на відрізку $[c,d]$ з кроком 0.001.

16. Написати програму, яка виводить всі числа Мерсена із заданого проміжку. Просте число називається числом Мерсена, якщо його можна представити у вигляді 2^p-1, де p — теж просте число.

17. Написати програму знаходження цілих чисел, які збільшаться на 20%, якщо їх цифри записати в зворотному порядку.

18. Знайти суму наступної послідовності $a_1+a_2-a_3+a_4-a_5+...+a_n$, де n – кількість елементів послідовності.

2.5. Контрольні запитання

1. Які форми запису має умовний оператор `if`?
2. Назвіть відмінні особливості операції умови в порівнянні з умовним оператором.
3. Які оператори використовуються для організації циклів в C++?
4. Які з циклів є циклами з передумовою, а які з постумовою?
5. Які три операції виконуються в циклі `for`?
6. Які керуючі оператори використовуються в циклах?

7. Що таке вкладені цикли?

Комп'ютерний практикум №3
Реалізація функцій

Мета роботи: отримати навички роботи з функціями.

3.1. Теоретичні відомості

Кожна програма у своєму складі повинна мати головну функцію `main()`. Саме функція `main()` забезпечує точку входу (початок виконання) в об'єктний модуль. Взагалі, функція — це іменована частина програмного коду, що, як правило, призначена для виконання визначених дій в межах програми.

Крім функції `main()`, в програму може входити будь-яка кількість функцій. Кожна функція по відношенню до іншої є зовнішньою. Для того, щоб функція була доступна, необхідно, щоб до її виклику про неї було відомо компілятору.

З поняттям функції у мові C++ пов'язано три наступних терміна:

- опис функції;
- прототип функції;
- виклик функції.

Опис функції складається з двох частин: заголовка та тіла. Опис функції має наступну форму запису:

```
/* заголовок функції*/
[тип_ результату] <ім'я>([список_параметрів])
{
  /* оголошення і оператори */
  тіло_функції
}
```

Тип результату — це тип значення, яке може повертати функція. При відсутності специфікатора типу передбачається, що функція повертає ціле значення (`int`). Якщо функція не повертає ніякого значення, то на місці типу записується специфікатор `void`. У списку параметрів для кожного параметра повинен бути зазначений тип. При відсутності параметрів список може бути порожнім або мати специфікатор `void`.

Тіло функції — це послідовність оголошень і операторів, які описують визначений алгоритм. Важливим оператором тіла функції є оператор

повернення в точку виклику: `return [вираз]`. Оператор `return` має подвійне призначення. Він забезпечує негайне повернення у зовнішню функцію і може використовуватися для передачі обчисленого у функції значення. У тілі функції може бути декілька операторів `return`, але може не бути й жодного. В останньому випадку повернення у місце виклику буде здійснено після виконання останнього оператора тіла функції.

Прототип функції може вказуватися до виклику функції замість опису функції для того, щоб компілятор міг виконати перевірку відповідності типів аргументів і параметрів. Прототип функції за синтаксисом подібний заголовку функції, але наприкінці його ставиться крапка з комою `;`. Параметри функції в прототипі можуть мати імена, які не є обов'язковими.

Компілятор використовує прототип функції для порівняння типів аргументів з типами параметрів. Мова C++ не передбачає автоматичного перетворення типів у випадках, коли аргументи не співпадають за типами з відповідними їм параметрами. Тобто мова C++ забезпечує строгий контроль типів.

Виклик функції може бути оформлений у вигляді

```
ім'я_функції(список_аргументів);
```

наприклад

```
f(x);
```

якщо у функції відсутнє значення, що повертається, або у вигляді виразу

```
h=f(x);
```

якщо таке значення існує. У наведених прикладах `f` — це ім'я функції, `x` — її фактичний параметр що передається при виклику, `h` — ім'я змінної, якій буде присвоєно значення, отримане за допомогою оператора `return`.

Кількість і типи формальних аргументів повинні співпадати з кількістю і типом фактичних параметрів функції. При виклику функції фактичні параметри підставляються замість формальних аргументів.

Передача аргументів за замовчуванням

Значення формальних параметрів можуть бути задані за замовчуванням. Зазвичай це константа, яка часто використовується при виклику функції. При цьому при описі функції перелік параметрів за замовчуванням повинен міститися в кінці списку формальних змінних функції. Нижче наведено декілька прикладів:

```
void foo(int i, int j = 7); // коректно
void foo(int i = 3, int j); // некоректно
void foo(int i, int j = 7, int k = 8); // коректно
```

```
void foo(int i = 1, int j = 7, int k = 8);
// коректно
void foo(int i, int j = 7, int k); // некоректно
```

Вбудовані та перевантажені функції

Зазвичай вибір імені функції повинен відобразити її основне призначення. Механізм перевантаження дозволяє використовувати одне ім'я для декількох функцій. Вибір конкретної реалізації залежить від типу аргументів функції. Наприклад:

```
int max(int, int);
double max(double, double);
```

Вбудовані функції в C++ визначаються за допомогою ключового слова `inline`. Воно розташовується перед оголошенням функції, коли необхідно, щоб код в тілі функції вбудовувався в місце виклику функції [1].

```
inline double cube (double x)
{
  return (x*x*x);
}
```

Обмеження компілятора не дозволяють вбудовувати складні функції. Крім того, ключове слово `inline` є лише рекомендацією компілятору.

Рекурсивні функції

В інженерній практиці доводитися часто реалізовувати рекурсивні алгоритми. Така необхідність виникає при роботі з динамічними структурами даних, таких як стеки, дерева, черги, тощо. Для реалізації рекурсивних алгоритмів у C++ передбачена можливість створення *рекурсивних функцій*. Рекурсивна функція — це функція, у тілі якої здійснюється виклик цієї ж функції.

Приклад 3.1. Програма для обчислення факторіала додатного числа

```
#include <iostream.h>

int fact(int n);

int main()
```

```
{
  int m;
  cout << "\nВведіть ціле число:";
  cin >> m;
  cout << "\n Факторіал числа " << m;
  cout << "дорівнює " << fact(m);
  return 0;
}

int fact(int n)
{
  int a;
  if (n<0) return 0;
  if (n==0) return 1;
  a =n * fact(n-1); // виклик цієї ж функції
    // для n-1
  return a;
}
```

Для від'ємного аргументу факторіала не існує, тому функція в цьому випадку повертає нульове значення. Оскільки за означенням факторіал нуля дорівнює 1, то в тілі функції передбачений і цей варіант. У випадку, коли аргумент функції `fact()` відмінний від 0 та 1, функцію `fact()` викликається із зменшеним на одиницю значенням параметра, а результат множиться на значення поточного параметра. Таким чином, в результаті вбудованих викликів функцій буде отримано наступний результат:

```
n * (n-l) * (n-2) * ... * 2 * 1 * 1
```

Приклад 3.2 Функція для обчислення максимума з двох чисел

```
#include<iostream.h>

int max(int, int); // прототип функції

void main()
{
  int x, y, z;
  cout << "\n почергово введіть x та y \n";
  cin >> x;
  cin >> y;
  z = max(x, y);
```

```
  cout << "z = " << z;
}

int max (int a, int b) // формальні аргументи
{
  int c; /* робоча змінна */
  if (a >= b)
    c = a;
  else
    c = b;
  return c;
}
```

3.2. Порядок виконання роботи

1. Проаналізувати умову задачі.
2. Розробити алгоритм та створити програму розв'язання задачі згідно з номером варіанту.
3. Результати роботи оформити протоколом.

3.3. Варіанти завдань

Обов'язкові завдання

1) $y = \frac{\sqrt{x-4}}{\sqrt{3a^3}}$;

2) $y = \log_4(x^2-4) + \frac{1}{x}$;

3) $y = \sqrt{\ln\left(\frac{1-a}{x}\right)}$;

4) $y = \frac{\sin(2x^2)}{x+2} + \cos\frac{x}{2a}$;

5) $y = \frac{\sqrt{x^2-4}}{2a\cos x}$;

15) $y = \cos x^3 + \frac{\sqrt{x}}{\sin a}$;

16) $y = tg\frac{1}{x-a} - \sqrt{2x+3a}$;

17) $y = x^3 + \frac{1}{\sqrt{a}}$;

18) $y = \frac{|x-b|}{2\sqrt{a}} - \frac{tgx}{b^2}$;

19) $y = \frac{x^2+x+3}{a-2} + \sqrt{\frac{(a-x)^5}{(2x-a)^3}}$;

6) $y = \dfrac{\sqrt{x+3} + x^3 + 3}{(x-1)^2} + \dfrac{\cos x^3}{2}$;

7) $y = \dfrac{\sqrt{x^2 - 7x} + \sin(2a)}{a-3}$;

8) $y = \sqrt{\log_3 \dfrac{(x-4)}{x+6}}$;

9) $y = \dfrac{\dfrac{2x^2}{z} + \sqrt{x-5}}{x+2} + \dfrac{x}{2z^3}$;

10) $y = \dfrac{\dfrac{x^3 + 2ax}{\sqrt{x+3}} + 3 + x3}{(x-1)^2}$;

11) $y = \sqrt{x^2 + 3} + \log_2 \dfrac{2a}{x-3}$;

12) $y = \sqrt{\dfrac{a-2}{2a+b} + \dfrac{\sin 3a}{\sqrt{\cos b}}}$;

13) $y = \dfrac{\left|\dfrac{5x}{x^3 + y^3}\right|}{\left|x^3\right| - 3} - ctg\dfrac{3x}{y}$;

14) $y = \dfrac{\sqrt{x^5 - y^5}}{\log_{10}(x+5)}$;

20) $y = \dfrac{\sqrt{x^2 + 4a} - \sqrt[3]{x}}{1 - \dfrac{x^2}{2}}$;

21) $y = \dfrac{\ln(x^3 - 8)}{\log_2(x^2 - 2)} + \dfrac{1}{\sin a}$;

22) $y = \sqrt{\dfrac{1}{x^3 - y^3}} - \dfrac{\sqrt{2x}}{x^3 - \dfrac{y}{2x}}$;

23) $y = \dfrac{\sin x}{\dfrac{2b}{\sqrt{x-b}}} - \dfrac{tgx}{b^2}$;

24) $y = \dfrac{x^5 + 2x^4}{z-2} + \dfrac{\sqrt{x-3}}{z-4}$;

25) $y = \sqrt{\dfrac{(a-b)^5}{\sqrt{a^2 + 2}}}$;

26) $y = \dfrac{\left|x^5 - \dfrac{z^5}{5}\right|}{\sqrt{x^2 - 9}}$;

27) $y = \dfrac{\sin(2x - a)}{\log_2(\frac{x}{a})}$;

28) $y = \dfrac{z3}{z+3} + \dfrac{\dfrac{x}{z^2}}{\left|\dfrac{x^2}{z-x}\right|}$;

Завдання підвищеної складності

1. Дано натуральне число *n*. Обчислити 1!+2!+3!+.....+*n*! Визначити функцію обчислення факторіала числа (факторіал числа *n* – це добуток всіх натуральних чисел від 1 до *n*, тобто *n*!=1×2×3×…×*n*).

2. Для заданих x та n обчислити значення виразу:

$$1+\frac{x}{1!}+\frac{x^2}{2!}+...+\frac{x^n}{n!}.$$

3. Для заданих x та n обчислити значення виразу:

$$\frac{1^2}{1+3^2}+\frac{2^2}{2+3^2}+\frac{3^2}{3+3^2}+...+\frac{n^2}{n+3^2}.$$

4. Для заданого n обчислити значення виразу

$$\frac{1}{1+3*3}+\frac{4}{2+3*3}+\frac{9}{3+3*3}+...+\frac{n*n}{n+3*3}.$$

5. Знайти всі прості числа із заданого інтервалу. Використайте функцію, яка визначає, чи є число простим.

6. Дано відрізки *a*, *b*, *c*, *d*. Для кожної трійки відрізків, із яких можна побудувати трикутник, знайдіть площу даного трикутника. Використайте функції визначення можливості побудови трикутника та обчислення площі.

7. Числа вводяться до тих пір, поки не буде введено перше від'ємне число. Визначити, скільки чисел із потоку введення рівні сумі кубів своїх цифр. Використайте функцію, яка буде перевіряти, чи дорівнює натуральне число сумі кубів своїх цифр.

8. Написати функцію, яка перевіряє, чи є число паліндромом (паліндром — це число, що читається однаково зліва направо та справа наліво, наприклад 12421).

9. Написати функцію, яка перевіряє, чи є число автоморфним (число автоморфне, якщо квадрат цього числа закінчується цим же числом, наприклад, числа 6 та 25, так як квадрати цих чисел 36 та 625).

10. Написати функцію, яка перевіряє, чи є число досконалим, тобто чи дорівнює воно сумі своїх дільників, за винятком самого себе.

11. Написати функцію знаходження суми цифр числа.

12. Написати програму пошуку найбільшого з трьох чисел, використовуючи функцію пошуку максимального з двох чисел.

13. Дано дійсні числа *x*, *y*. Обчислити $\frac{f(x^2, y, x^2 y)}{f(x - y, x^2, y)}$, де

$$f(a,b,c)=\frac{\sqrt{2a-b}}{5+\sqrt{c}}.$$

14. Дано дійсні числа x, y. Обчислити $g(x, y) + g(1.5x, y^2) - g(2x, 5y)$, де

$$g(a,b) = \frac{\sum_{i=0}^{5} \frac{a^i + b^i}{ai + 2ab + 4i^2}}{\sum_{i=0}^{5} \frac{a^{2i} + b^{2i}}{i}}.$$

15. Дано дійсні числа x, y. Обчислити $\frac{1.8g(x, y) + g(1.3x, x - y)}{g(y^3)}$, де

$$g(a,b) = \frac{\sum_{i=0}^{5} \frac{a^i - b^i}{i!}}{\sum_{i=0}^{5} \frac{2a^{2i} + 3b^{3i}}{(5i)!}}.$$

16. Дано дійсні числа x, y. Обчислити $t(x,5) + t(y,5) + \max(t(x^5 - y^5, 5), x^5)$, де $t(a,b) = (a-b)^2 + \frac{a^2}{b^2}$.

Використайте функцію знаходження максимального з двох чисел.

17. Дано дійсні числа x, y. Обчислити $t(x^2 + y^3, x - y) + t(x + y, 3 + x) + \min(y, \sqrt{t(x^2 - y, 2)})$, де $t(a,b) = \sqrt{a-b} + \frac{a^2 + \sqrt{a}}{\sqrt{b} + b^2}$. Використати функцію знаходження мінімального з двох чисел.

18. Дано дійсні числа x, y. Обчислити

$$t = \begin{cases} \frac{x^2 + y^3}{x + 2y}, & \text{якщо } \min(y, \sqrt{x^2 - y}) < 0 \\ \frac{x^2 + y^3 + \frac{1}{x}}{x + 2y}, & \text{якщо } \max(\sqrt{y}, x^2 - y) < 0 \end{cases}.$$

Використати функції знаходження мінімального та максимального із двох чисел.

19. Для біноміальних коефіцієнтів (число перестановок з n по k) виконується наступне співвідношення: $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$, $C_n^0 = C_n^n = 1$. Реалізувати функцію для обчислення значення C_n^k, використовуючи це рекурсивне співвідношення.

20. Числа Фібоначчі f_n обчислюються за формулами $f_0=f_1=1$; $f_n=f_{n-1}+f_{n-2}$ при $n=2,3,...$ Реалізувати функцію, яка за заданим номером n обчислюватиме значення f_n: (а) з використанням рекурсії; (б) з використанням «довгої арифметики» для $n<65536$.

21. Реалізувати за допомогою рекурсивної функції алгоритм quicksort («швидке сортування») для сортування елементів масиву.

22. Реалізувати рекурсивну функцію для розв'язання задачі про Ханойські вежі.

23. Реалізувати рекурсивну функцію для обчислення найбільшого спільного дільника двох натуральних чисел A та B, використовуючи алгоритм Евкліда.

3.4. Контрольні запитання

1. Поясніть значення прототипу функції.
2. Який зв'язок між параметрами функції і аргументами? Поясніть механізм передачі аргументів по значенню.
3. Яким чином можна забезпечити отримання за допомогою функції декількох результатів?
4. Які змінні називаються глобальними?
5. У чому відмінність глобальних та локальних змінних?
6. Поясніть призначення різних класів пам'яті.
7. Наведіть означення рекурсивної функції та приклад їх застосування.
8. Наведіть приклад перевантаження функції.
9. У чому перевага використання вбудованих функцій?
10. Які правила використання аргументів за замовчуванням

Комп’ютерний практикум №4
Робота з пам’яттю

Мета роботи: отримати навички роботи з вказівниками, посиланнями та динамічними масивами.

4.1. Теоретичні відомості

Вказівники

У мові C++ є операція визначення адреси — `&`, за допомогою якої визначаються адреса комірки пам’яті, що містить задану змінну. Наприклад, якщо `vr` — ім’я змінної, то `&vr` — адреса цієї змінної. У C++ також існують і змінні типу вказівник.

Вказівник — це похідний тип даних, який використовується для зберігання адрес змінних і об'єктів. Значенням змінної-вказівника є адреса змінної або об'єкта. Опис таких змінних здійснюється за допомогою наступних виразів:

```
<тип> *<ім'я вказівника на змінну заданого типу>;
```

Приклади опису вказівників:

```
int *ptri;  //вказівник на змінну цілого типу
char *ptrc; //вказівник на змінну символьного типу
float *ptrf;//вказівник на змінну з плаваючою
            // крапкою
```

Змінні різних типів займають різну кількість комірок пам'яті. При цьому для деяких операцій з вказівниками необхідно знати об'єм відведеної пам'яті. Однак самі змінні типу вказівник мають однаковий розмір.

Нехай змінна-вказівник має ім'я `ptr` (тобто оголошена як `int* ptr`), тоді в якості значення їй можна присвоїти адресу за допомогою наступного оператора:

```
ptr=&vr;
```

Наприклад (рис. 4.1):

```
int vr = 1;
int* ptr = &vr; // ptr містить адресу змінної vr
```

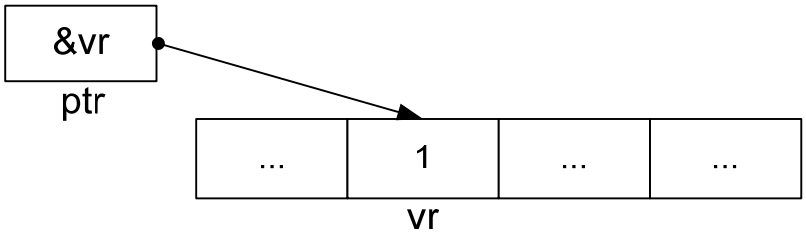


Рис. 4.1.

У мові C++ при роботі з вказівниками велике значення має операція непрямої адресації (**розіменування вказівника**) — `*`. Операція `*` дозволяє звертатися до змінної не напряму, а через вказівник, який містить адресу цієї змінної. Ця операція є одномісною і має асоціативність зліва на право. Цю операцію не слід плутати з бінарною операцією множення. Нехай `ptr` — вказівник, тоді `*ptr` — це значення змінної, на яку вказує `ptr`. Для вищенаведеного прикладу:

```
int *ptr;  // оголошення змінної типу вказівник
*ptr=1;    // розіменування вказівника (значення
           //змінної vr, на яку вказує вказівник)
```

Операція `*` у деякому розумінні є оберненою до операції `&`. Розглянемо приклад роботи з вказівниками та посиланнями [1]:

```
int a = 1, b;
int* ptr = &a; //містить адресу змінної a
cout << " Змінна a = " << (*ptr);
cout << "зберігається за адресою " << ptr;

b = ptr;
//помилка: вказівник не перетворюється у ціле число
b = *ptr + 1; // b = 2
```

Посилання

Посилання (reference) — це видозмінена форма вказівника, що може використовуватись як псевдонім (інше ім'я) змінної. Тому посилання не

потребують додаткової пам'яті. Для визначення посилання використовують символ & (амперсант), який ставиться перед змінною-посиланням.

Більш докладно посилання розглядаються у наступному семестрі.

Приклад 4.1. Використання посилань

```
#include <iostream>
using namespace std;
int main()
{
  int t = 13,
  int &r = t;// ініціалізація посилання на t
             // тепер r синонім імені t
  cout << "Початкове значення t:" << t;
      // виводить 13
  r += 10; // зміна значення t через посилання
  cout<<"\n Остаточне значення t:" << t;
  // виводить 23
  return 0;
}
```

У даному випадку посилання використовувалось в якості псевдоніму змінної. У цій ситуації воно називається незалежним посиланням (*independent reference*) і повинно бути ініціалізоване на момент оголошення. Такий спосіб використання посилань може призвести до фатальних помилок, які важко виявити через виникнення плутанини у використанні змінних.

Масиви та вказівники

Масив — це набір однотипних об'єктів, які мають спільне ім'я і відрізняються місцезнаходженням в цьому наборі (або індексом, присвоєним кожному елементу масиву). Елементи масиву займають одну неперервну область оперативної пам'яті комп'ютера і розміщені послідовно один за одним, починаючи з базової адреси.

Нижче наведено декілька прикладів одновимірних масивів.

```
int masl[492];// масив з 492 елементів,
//оголошений як глобальна змінна
int main()
{
double mas2[250];// масив з 250 чисел типу double
static char mas3[20];
//статичний рядок з 20 символів
int mas4[2][4];
// двовимірний масив з чисел типу int
...
}
```

В наведеному прикладі квадратні дужки `[]` означають, що всі ідентифікатори, після яких вони стоять, є іменами масивів. Число в дужках визначає кількість елементів масиву. Доступ до окремого елементу масиву здійснюється з використанням номера цього елементу, або індексу. Нумерація елементів масиву починається з нуля і закінчується n-1, де n — кількість елементів масиву.

Ініціалізація масиву — це присвоєння початкових значень його елементам при оголошенні. Масиви можна ініціалізувати списком значень або відокремлених комою виразів, розташованих у фігурних дужках.

Ініціалізацію масиву, елементи якого містять кількість днів в кожному місяці року, можна виконати наступним чином:

```
int days[12]={31,28,31,30,31,30,31,31,30,31,30,31};
```

Якщо список значень ініціалізації коротший за довжину масиву, то проініціалізованими будуть перші елементи масиву, а решта ініціалізуються нулем.

Масив можна також ініціалізувати списком без зазначення в дужках довжини масиву. При цьому масиву присвоюється довжина за кількістю ініціалізаторів. Наприклад,

```
char code[] = {'a', 'b', 'c'};
```

В даному прикладі масив `code` буде мати довжину 3.

Автоматичні масиви (оголошені в блоці) нічим не ініціалізуються і містять невідому інформацію.

Взагалі кажучи, ім'я масиву є *константним вказівником*, який ініціалізовано базовою адресою [1]. Таким чином, масиви і вказівники використовуються для однієї мети: доступу до пам'яті. Різниця полягає в тому, що вказівник є змінною, яка приймає в якості значення адресу комірки пам'яті. А ім'я масиву може розглядатися як константний вказівник з фіксованою (базовою) адресою [1].

```
int mas[4];
int* ptr = mas;
```

Табл. 4.1

Адреса	`&mas[0]` або `ptr`	`&mas[1]` або `ptr+1`	`&mas[2]` або `ptr+2`	`&mas[3]` або `ptr+3`
Значення	`mas[0]` або `*ptr`	`mas[1]` або `*(ptr+1)`	`mas[2]` або `*(ptr+2)`	`mas[3]` або `*(ptr+3)`

Таким чином, в даному випадку записи `ptr+i` та `&mas[i]` рівносильні, де `i` — деяке зміщення (ціле число) від адреси `ptr` або `&mas` (в даному випадку `i`=0..3). Відмінність полягає в тому, що значення `ptr+i` змінювати можна, а `&mas[i]` — ні. Наступні вирази є некоректними [1]:

```
mas = ptr;
++mas;
mas = mas + 3;
```

Для отримання значення елементу масиву через вказівник `ptr` необхідно використати операцію розіменування. Розглянемо два способи знаходження суми елементів деякого масиву [1]:

```
const int N = 10;
int mas[N];
int* ptr = mas; // або ptr = &mas[0]

// 1 варіант: через вказівник ptr
int sum1 = 0;
for (ptr = mas; ptr < &mas[N]; ++ptr)
   sum1 += *ptr;

// 2 варіант: з використанням індексів
int sum2 = 0;
```

```
for (int i = 0; i < N; ++i)
   sum2 += mas[i];
   // рівносильно sum2 += *(mas + i);
```

Аналогічна адресна арифметика використовується при використанні масивів більшої розмірності. Розглянемо оголошення двовимірного масиву:

```
int mas[4][2]; // матриця розміром 4 на 2
int *ptr;
```

Тоді вираз `ptr=mas` вказує на перший стовпець першого рядка матриці `mas`. Записи `mas` і `&mas[0][0]` рівносильні. Тоді вираз `ptr+1` вказуватиме на `mas[0][1]`, далі йдуть елементи: `mas[1][0]`, `mas[1][1]`, `mas[2][0]` і т. д.; `ptr+5` вказуватиме на `mas[2][1]`. Тобто двовимірні масиви розташовані в пам'яті так само, як і одновимірні масиви, займаючи послідовні комірки пам'яті:

Табл. 4.2

Адреса	ptr	ptr+1	ptr+2	ptr+3	ptr+4	ptr+5	...
Значення	mas[0][0]	mas[0][1]	mas[1][0]	mas[1][1]	mas[2][0]	mas[2][1]	...

Слід зауважити, що розмірність статичного масиву є константою і не може визначатися під час виконання програми.

Динамічні масиви

Динамічним називається масив, розмірність якого стає відомою в процесі виконання програми.

В C++ для роботи з динамічними об'єктами використовують спеціальні оператори `new` та `delete`. Ці оператори використовуються для керування вільною пам'яттю. Вільна пам'ять (або куча, heap) — це область пам'яті, яка надається системою для розміщення об'єктів, час життя яких напряму керується програмістом [1]. За допомогою оператора `new` виділяється пам'ять під динамічний об'єкт (який створюється в процесі виконання

програми), а за допомогою оператора `delete` створений об'єкт видаляється з пам'яті. Оператора `new` має наступнимй синтаксис.

```
new ім'я_типу;
new ім'я_типу ініціалізатор;
new ім'я_типу[вираз];
```

В результаті виконання оператору `new` в пам'яті виділяється об'єм пам'яті, який необхідний для зберігання вказаного типу, і повертається базова адреса. Якщо пам'ять недоступна, оператор `new` повертає значення `0`, або генерує виключення.

Оператор `delete` має наступний формат:

```
delete вираз;
delete[] вираз;
```

Розглянемо виділення пам'яті під динамічний масив. Нехай розмірність динамічного масиву вводиться з клавіатури. Спочатку необхідно виділити пам'ять під цей масив, а потім створений динамічний масив необхідно вилучити з пам'яті. Це можна зробити наступним чином.

```
int n; // n – розмірність масиву
cin >> n; // вводимо з клавіатури
int* mas = new int[n];
// виділення пам'яті під динамічний масив
delete[] mas; // звільнення пам'яті
```

В цьому прикладі змінна `mas` є вказівником на масив з `n` елементів. Вираз `int* mas = new int[n]` виконує дві дії: оголошується змінна типу вказівника на `int`, далі вказівнику надається адреса виділеної області пам'яті у відповідності з заданим типом об'єкта.

Для цього ж прикладу можна задати наступну еквівалентну послідовність операторів:

```
int n, *mas; // n - розмірність масиву, mas –
вказівник на тип int
cin >> n;
mas = new int[n]; // виділення пам'яті під масив
delete[] mas; // звільнення пам'яті
```

Оператор `delete[] mas` використовується для звільнення виділеної пам'яті.

Зауваження! Завжди використовуйте оператор `delete` після виділення пам'яті за допомогою оператора `new`. Це входить в обов'язки програміста! Інакше це може призвести до втрати пам'яті.

Приклад 4.2

Використовуючи вказівники, вивести на екран масив із заданою кількістю елементів

```
#include <iostream>
using namespace std;
int main()
{
int *a;
int n;
cout<<"Enter number of elements in your massive:";
cout <<endl;
cin>>n;
cout<<"Your massive: "<<endl;
a=new int [n];
for (int i=0;i<n;i++)
{
   *(a+i)=i+1;
   cout<<*(a+i)<<endl;
}
delete []a;
cin.get();
cin.get();
return 0;
}
```

Програма може мати і такий вигляд:

```
#include <iostream>
using namespace std;
int main()
{
int *a;
int n;
cout<<"Enter number of elements in your massive:";
cout <<endl;
```

```
cin>>n;
cout<<"Your massive: "<<endl;
a=new int [n];
for (int i=0;i<n;i++)
{
   a[i]=i+1;
   cout<<a[i]<<endl;
}
delete []a;
cin.get();
cin.get();
return 0;
}
```

Приклад 4.3. Створення динамічного двовимірного масиву

```
#include <iostream>
using namespace std;

int main()
{
   int n, m;
   cout << "Введіть кількість рядків";
   cin >> n;
   cout << "Введіть кількість стовпців";
   cin >> m;

   int** a; //a - вказівник на масив вказівників
   a = new int*[n];
   //виділення пам'яті для масиву вказівників на
   //n рядків
   for(int i = 0; i < n; i++)
       a[i] = new int[m];
   //виділення пам'яті для кожного рядка масиву
   // розмірністю m
...
// Вивід елементів масиву
```

```
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cout << a[i][j] << "   ";
        }
        cout<<endl;
    }

// Видалення пам'яті
    for(int i = 0; i < n; i++)
        delete[] a[i];
        //звільнення пам'яті від кожного рядка
    delete[] a;
    //звільнення пам'яті від масиву вказівників
    return 0;
}
```

Розглянемо цей приклад більш детально. Спочатку створюється подвійний вказівник `int** a`: вказівник на масив вказівників. Під цей масив вказівників пам'ять виділяється за допомогою оператора `new`: `a = new int*[n]` (див. рис.). Потім для кожного такого вказівника (їх кількість становить `n`) створюється окремий динамічний одновимірний масив розмірності `m`:

```
for(int i = 0; i < n; i++)
    a[i] = new int[m];
```

Таким чином, ми отримаємо матрицю розміром $n \times m$.

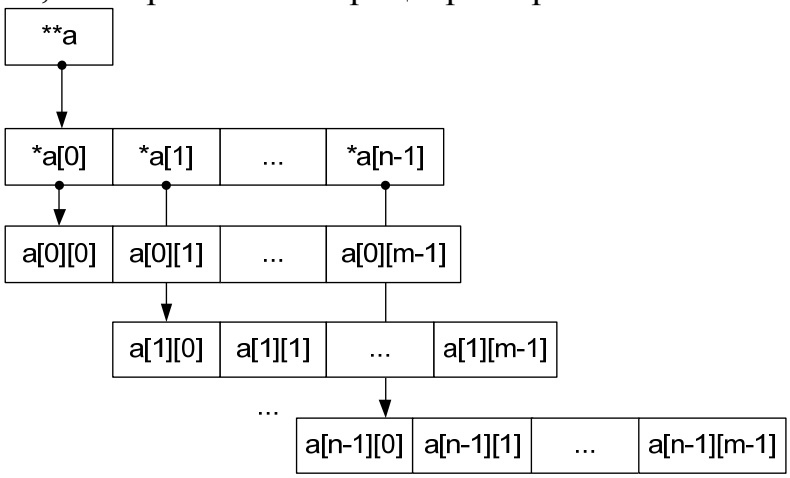


Рис. 4.2.

Передача масивів у функції

Одновимірні масиви

Масиви можуть бути параметрами функцій, і функції в якості результату можуть повертати вказівник на масив. Розглянемо ці можливості.

Якщо в якості параметра функції передається лише ім'я масиву, то виникає необхідність визначення в тілі функції кількості його елементів. Тому при передачі масиву у функцію необхідно передавати у функцію не лише ім'я масиву, а й кількість його елементів. Так само, якщо функція повинна повертати масив, то краще це робити теж через параметри. А саме, визначити ще пару параметрів – вказівник на перший елемент результуючого масиву (ім'я масиву) і його розмірність.

При роботі з *рядками* — масивами типу `char[]` — проблема розв'язується просто, оскільки останній елемент рядка має значення `\0` (нульовий символ). Тому при обробці масиву-аргументу кожен його елемент аналізується на наявність символу кінця рядка.

Приклад 4.4. Рядок як параметр функції

Потрібно скласти програму, що містить функцію, яка підраховує кількість елементів у рядку. Можливий варіант програми має вид:

```
//Масиви у функціях
#include <iostream>;
using namespace std;
int dl(char[]);//прототип функції dl
int main()
{
char p[]="кафедра";
cout << "\n Довжина рядка дорівнює:" << dl(p);
return 0;
}

int dl(char c[])
{
int i;
for(i=0;;i++)
if (c[i]=='\0') break;
```

```
return i;
}
```

У результаті виконання програми на екран буде видано повідомлення:

```
Довжина рядка дорівнює: 7.
```

Якщо масив-параметр функції не є *символьним рядком*, то необхідно або використовувати масиви з заздалегідь відомим числом елементів, або передавати розмір масиву за допомогою додаткового параметра. Наступні два приклади ілюструють ці можливості.

Приклад 4.5. Одномірний масив як параметр функції

Нехай потрібно скласти програму, у якій функція обчислює суму елементів масиву, що складає з 5 елементів. Можливий варіант програми має вид:

```
// Одномірний масив як параметр
#include <iostream>
using namespace std;

int sum(float x[5])
{
float s=0;
for(int i=0;i<5;i++) s=s+x[i];
return s;
}
int main()
{
float z[5], y;
for(int i=0;i<5;i++)
{
   cout<<"\n Введіть черговий елемент масиву:";
   cin >> z[i];
}
y=sum(z);
cout<<"\n Сума елементів масиву:”<< y;
return 0;
}
```

У результаті виконання програми на екран буде виведене значення суми елементів масиву `z`.

У приведеному прикладі заздалегідь відоме число елементів — 5, тому у функції всього один параметр — масив `x`. Оскільки у функції існує значення, що повертається, то виклик функції може бути тільки виразом чи частиною виразу. У наведеній програмі такий вираз міститься у правій частині. Тут аргументом функції є масив `z`.

Багатовимірні масиви

За іменем масиву неможливо довідатися його розмірність. При передачі масивів як параметрів функції варто враховувати, що передавати масиви можна тільки з однією невизначеною границею мірності (ця мірність повинна бути самою лівою).

Приклад 4.6. Використання двовимірного масиву як параметру функції

Нижче наведено програму з функцією, що підраховує суму елементів матриці.

```
#include <iostream>
using namespace std;

float summa(int n,float a[][3])
{
  float s=0;
  for(int i=0;i<n;i++)
    for(int j=0;j<3;j++)
      s=s+a[i][j];
  return s;
}
void main()
{
  float z[4][3]={0,1,2,3,4,5,6,7,7,6,5,4 };
  cout<< "\n Сума елементів матриці дорівнює";
  cout << summa(4,z);
}
```

У результаті виконання програми на екран буде виведене повідомлення: `Сума елементів матриці дорівнює 50`.

4.2. Порядок виконання роботи

1. Проаналізувати умову задачі.
2. Розробити алгоритм та створити програму розв'язання задачі згідно з номером варіанту.
3. Результати роботи оформити протоколом.

4.3. Варіанти завдань

Обов'язкові завдання

Реалізувати наступні завдання з використанням статичних та динамічних масивів. Для доступу до елементів динамічного масиву використати два підходи: через явне розіменування вказівника та через індекси.

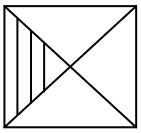
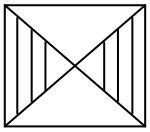
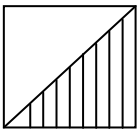
1. Знайти мінімальний елемент матриці заданого розміру та добуток всіх її додатних елементів.
2. Знайти скалярний добуток двох векторів та максимальний елемент кожного з них.
3. Обчислити добуток матриці на вектор та максимальний елемент отриманого вектора.
4. Знайти добуток двох матриць та мінімальний елемент отриманої матриці.
5. Знайти суму двох матриць та обчислити слід (суму діагональних елементів) отриманої матриці.
6. Дано два однакові за довжиною одновимірні масиви. Об'єднати їх у третій масив, чергуючи елементи першого та другого масивів.
7. Дано одновимірний масив дійсних чисел. Визначити, кількість додатних, від'ємних та нульових елементів.

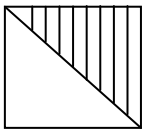
Завдання підвищеної складності

1. Заповнити двовимірний масив розмірності *n*x*m* за заданим правилом:	1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 0 10 0

2. Заповнити двовимірний масив розмірності *n*x*n* за заданим правилом:	0 1 1 1 1 0 2 0 1 1 0 4 2 2 0 0 4 4 2 2 0 0 4 4 2 0 3 3 0 4 0 3 3 3 3 0
3. Заповнити двовимірний масив розмірності *n*x*n* за заданим правилом:	6 1 1 1 1 5 2 6 1 1 5 4 2 2 6 5 4 4 2 2 5 6 4 4 2 5 3 3 6 4 5 3 3 3 3 6
4. Заповнити двовимірний масив розмірності *n*x*m* за заданим правилом:	1 2 3 4 5 6 12 11 10 9 8 7 13 14 15 16 17 18 24 23 22 21 20 19
5. Заповнити двовимірний масив розмірності *n*x*n* наступним чином: перший рядок - числа Фібоначчі, а кожний стовпець продовжує ряд Фібоначчі від елемента, що знаходиться в першому рядку (числа Фібоначчі f_n обчислюються за формулами $f_0=f_1=1$; $f_n=f_{n-1}+f_{n-2}$ при $n=2,3,...$ Числова послідовність Фібоначчі 1, 1, 2, 3, 5, 8, 13, ...).	
6. Заповнити двовимірний масив розмірності *n*x*n* наступним чином: перший рядок та стовпець масиву заповнені одиницями, а кожний інший елемент дорівнює сумі сусідів зверху та зліва.	
7. Дано двовимірний масив розмірності *n*x*n*. У рядках з від'ємним елементом на головній діагоналі знайти суму всіх елементів і найбільший із всіх елементів.	

8. Заповнити двовимірний масив розмірності *n*x*m* за заданим правилом:	1 8 9 16 17 24 2 7 10 15 18 23 3 6 11 14 19 22 4 5 12 13 20 21
9. Заповнити двовимірний масив розмірності *n*x*m* за заданим правилом:	1 5 9 13 17 2 6 10 14 18 3 7 11 15 19 4 8 12 16 20
10. Заповнити двовимірний масив розмірності *n*x*n* так, щоб на головній діагоналі були розміщені числа від *N* до 1, під головною діагоналлю нулі, а над головною діагоналлю по рядках числа в порядку зростання від заданого.	
11. Заповнити двовимірний масив розмірності *n*x*m* за заданим правилом:	19 20 21 22 23 24 18 17 16 15 14 13 7 8 9 10 11 12 6 5 4 3 2 1
12. Дано квадратну матрицю розмірності *n*x*n*. Отримати транспоновану матрицю (транспонуванням квадратної матриці називається таке перетворення, при якому рядки та стовпці міняються місцями: *i*-й стовпець стає *i*-им рядком).	
13. Заповнити двовимірний масив розмірності *n*x*n* за заданим правилом:	1 3 4 10 11 2 5 9 12 19 6 8 13 18 20 7 14 17 21 24 15 16 22 23 25
14. Дано двовимірний масив розмірності *n*x*m*. Визначити, чи є в масиві рядки, рівні першому. Якщо є, вивести їхні індекси.	

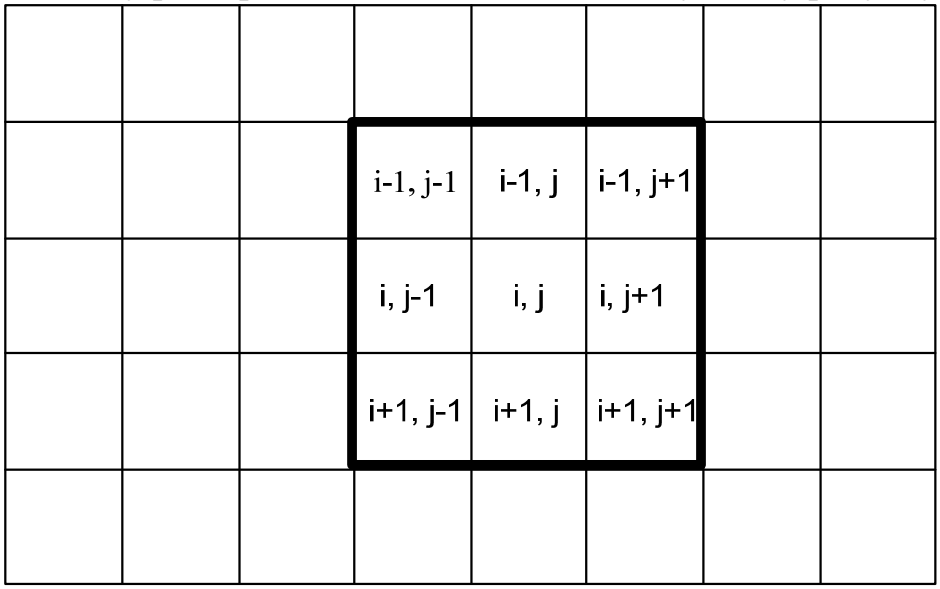
15. Заповнити двовимірний масив розмірності *n*x*m* з клавіатури лише невід'ємними числами, передбачивши захист елементів цього масиву від неправильного введення. Знайти кількість нульових елементів, розташованих у непарних рядках.	
16. Заповнити двовимірний масив розмірності *n*x*n* одиницями в шаховому порядку, починаючи з верхнього лівого кута.	
17. Дано квадратну матрицю розмірності *n*x*n*. Знайти суму елементів та максимальний елемент у заштрихованій області.	
18. Для даного двовимірного масиву розмірності *n*x*m* знайти середнє арифметичне найбільшого і найменшого значень його елементів. Замінити цим значенням всі елементи заданого рядка.	
19. Дано квадратну матрицю розмірності *n*x*n*. Знайти суму елементів та максимальний елемент у заштрихованій області.	
20. Дано двовимірний масив розмірності *n*x*m*. Знайти номери рядків, елементи в кожному з яких однакові між собою.	
21. Заповнити двовимірний масив розмірності *n*x*m* з клавіатури лише простими числами, передбачивши захист елементів цього масиву від неправильного введення. Знайти суму елементів, які мають непарну суму індексів.	
22. Дано квадратну матрицю розмірності *n*x*n*. Знайти суму елементів та максимальний елемент у заштрихованій області.	

23. Заповнити двовимірний масив розмірності *n*x*m* з клавіатури лише числами кратними 3, передбачивши захист елементів цього масиву від неправильного введення. Знайти суму тих елементів масиву, які без остачі діляться на 9.	
24. Дано двовимірний масив розмірності *n*x*m*. Знайти номери рядків, всі елементи в яких парні.	
25. Дано двовимірний масив розмірності *n*x*n*. Обчислити суму тих його елементів, які розташовані на головній діагоналі і вище неї та перевищують по величині всі елементи, які розташовані нижче головної діагоналі. Якщо на головній діагоналі і вище неї немає елементів із зазначеною властивістю, то видати відповідне повідомлення.	
26. Дано двовимірний масив розмірності *n*x*m*. Знайти суму найбільших значень елементів його рядків.	
27. Дано квадратну матрицю розмірності *n*x*n*. Знайти суму елементів та максимальний елемент у заштрихованій області.	
28. «Магічним» квадратом називається квадратна матриця цілих чисел від 1 до *N*, розташованих так, що суми елементів кожного рядка, кожного стовпця та обох діагоналей однакові і рівні (1/2)**N*(1+*N*). Побудувати «магічний» квадрат.	
29. Побудувати і вивести на екран "латинський" квадрат - масив, що складається з *n* різних чисел, всіх по *n* раз розташованих так, що в кожному рядку й стовпці кожне число зустрічається лише один раз.	
30. Дано двовимірний масив розмірності *n*x*m*. Знайти рядок з найбільшою сумою елементів і найменшою. Вивести знайдені рядки і суми їхніх елементів.	

Завдання високої складності

1. Нехай дано матрицю, кожний елемент якої інтерпретується як значення інтенсивності пікселя деякого зображення. Необхідно обробити дану матрицю (зображення) з використанням медіанного фільтру заданого розміру: розмір околу для кожного елемента (пікселя) може складати 3 на 3, 5 на 5, 7 на 7, тощо. Медіаною набору елементів $\{a, b, c\}$ при умові $a < b < c$ є значення b, а набору $\{a, b, c, d\}$ при умові $a < b < c < d$ — значення $(b + c)/2$. Реалізувати медіанний фільтр для різних розмірів околу. Перевірити розроблений алгоритм для матриці невеликого розміру (наприклад, 20 на 20) та матриці розміром 2000 на 2000. Значення елементів матриці можуть бути вибрані випадковим чином з проміжку 1...255.

Приклад околу розміром 3 на 3 показано на наступному рисунку:



Значення елементу `mas[i][j]` змінюється на медіану з набору значень `{mas[i-1][j-1], mas[i-1][j], mas[i-1][j+1], mas[i][j-1], mas[i][j], mas[i][j+1], mas[i+1][j-1], mas[i+1][j], mas[i+1][j+1]}`.

2. Нехай дано матрицю, кожний елемент якої інтерпретується як значення інтенсивності пікселя деякого зображення. Необхідно знайти градієнт цього зображення (так званий оператор Собеля).

Для елементу масиву `mas[i][j]` градієнт можна оцінити наступним чином:

Градієнт (похідна) по горизонталі:

$$G_x[i][j] = \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \otimes \begin{pmatrix} mas[i-1][j-1] & mas[i-1][j] & mas[i-1][j+1] \\ mas[i][j-1] & mas[i][j] & mas[i][j+1] \\ mas[i+1][j-1] & mas[i+1][j] & mas[i+1][j+1] \end{pmatrix}$$

= `mas[i-1][j-1] + 2*mas[i-1][j] + mas[i-1][j+1] - mas[i+1][j-1] - 2*mas[i+1][j] - mas[i+1][j+1]`.

(Операції ⊗ в даному випадку визначає по елементний добуток).

Градієнт (похідна) по горизонталі:

$$G_y[i][j] = \begin{pmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} mas[i-1][j-1] & mas[i-1][j] & mas[i-1][j+1] \\ mas[i][j-1] & mas[i][j] & mas[i][j+1] \\ mas[i+1][j-1] & mas[i+1][j] & mas[i+1][j+1] \end{pmatrix}$$

= `mas[i-1][j-1] + 2*mas[i][j-1] + mas[i+1][j-1] - mas[i-1][j+1] - 2*mas[i][j+1] - mas[i+1][j+1]`.

Тоді градієнт для кожного елементу `[i, j]` становить:

$$G[i][j] = \sqrt{\left(G_y[i][j]\right)^2 + \left(G_y[i][j]\right)^2}.$$

Перевірити розроблений алгоритм для матриці невеликого розміру (наприклад, 20 на 20) та матриці розміром 2000 на 2000. Значення елементів матриці можуть бути вибрані випадковим чином з проміжку 1...255.

4.4. Контрольні запитання

1. Що таке вказівник? Як він використовується? Який сенс операції розіменування?
2. Як оголошується змінна типу вказівник?
3. Чому при оголошенні вказівника необхідно вказувати тип змінної, яка адресується за його допомогою?
4. Що таке посилання? Яка область застосувань посилань? Яке значення операції &?
5. Як співвідносяться вказівники та посилання?
6. Порівняйте різні способи передачі аргументів в функції? Відзначте переваги та недоліки.
7. Як співвідносяться масиви та вказівники? Що розуміється під терміном «постійний вказівник»?

8. Наведіть приклади різних способів доступу до елементів масиву?
9. Що таке динамічний масив? Яка відмінність від статичного?
10. Яке застосування операторів new і delete? Що є результатом виконання цих операторів?
11. Які способи передачі масивів у функції Ви знаєте? В яких випадках виникають некоректності і як їх розв'язувати?
12. Що таке вказівник на функцію? Яким чином можна викликати функції через вказівник? Наведіть області застосування вказівників на функцію.

Комп'ютерний практикум №5
Робота з файлами

Мета роботи: отримати навички роботи з файлами з використанням засобів мови С++.

5.1. Теоретичні відомості

У мові С/С++ всі операції введення/виведення реалізуються за допомогою бібліотечних функцій, що входять до складу конкретної системи програмування. Під час роботи з файлами дані можуть передаватися у бінарному або в текстовому форматі.

Бібліотека С/С++ підтримує два основних рівні роботи з файлами: потокове введення/виведення (заголовний файл `fstream`) та форматоване введення/виведення за допомогою функцій (заголовний файл `stdio.h`).

Мова С є фундаментом С++. При цьому С++ підтримує всі засоби роботи з файлами мови С. Тому при використанні С-коду в програмі на мові С++ немає необхідності змінювати процедури введення/виведення. В той же час слід зазначити, що при написанні програм на С++ зазвичай більш зручно використовувати саме засоби С++. З точки зору засобів програмування С/С++ файл є іменованим місцем на жорсткому диску, що у загальному випадку може бути пов'язаний з реальним фізичним пристроєм (рис. 5.1).

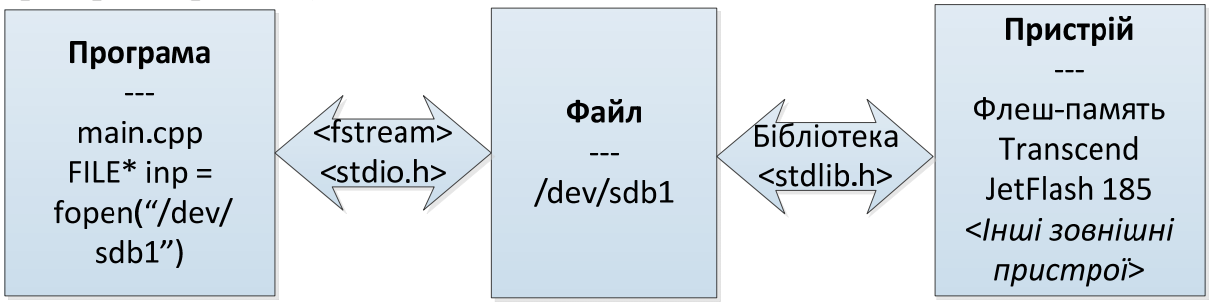


Рис. 5.1. Використання файлів в програмі С/С++

Файл від'єднується від програми за допомогою операції закриття файлу. З фізичної точки зору для забезпечення зв'язку між файлом та програмою створюється потік, через який і передаються (зчитуються) дані. При закритті файлу, відкритого для виведення, дані пов'язаного з ним потоку

записуються на зовнішній пристрій. Цей процес називають *дозаписом потоку*. При цьому гарантується, що ніяка інформація випадково не залишиться в буфері потоку.

Якщо програма завершує роботу нормально, або функція `main()` повертає керування операційній системі, або вихід відбувається шляхом виклику функції `exit()`, всі файли закриваються автоматично.

У випадку аварійного завершення роботи програми, наприклад, її краху або завершення шляхом виклику функції `abort()`, файли автоматично закриватися не будуть. Це може призвести до втрати даних.

При використанні для роботи з файлами бібліотечних функцій `<stdio.h>` використовується спеціальна керуюча структура, що містить інформацію про файл та надає тимчасовий буфер для зберігання даних. Вона має тип `FILE`. Крім тимчасового буферу у керуючій структурі міститься інформація про ідентифікатор файлу, його розташування на диску та покажчик поточної позиції у файлі.

Більш сучасним підходом до роботи з файлами є потоки C++. Їх можливості в повній мірі розкриваються при використанні об'єктного підходу. Детальну інформацію щодо переваг потоків C++ Ви отримаєте з курсу **Програмування. Об'єктно-орієнтований підхід** та додаткової літератури.

Бібліотечні функції `<stdio.h>` мають наступні переваги:

1. Зручні можливості форматованого введення/виведення.
2. Забезпечують більшу швидкість передачі даних.

та недолік:

1. Контроль типів повинен забезпечуватись розробником програми.

Основною перевагою потоків C++ є автоматичний контроль типів.

При всьому різноманітті засобів обох підходів існує типовий сценарій роботи з файлами, а саме наступний:

- **Відкриття файлу.** При відкритті файлу вказується ім'я файлу, визначається режим доступу (читання, запис) та тип файлу (текстовий або двійковий).
- **Читання або запис даних**. Після успішного відкриття файлу з нього можна прочитати або записати дані у визначеному форматі (форматоване введення/виведення). Наприклад, в файл можна записати значення змінної *a* в шістнадцятковій системі числення в полі розміром 10 символів з вирівнюванням вліво.
- **Закриття файлу**. Для завершення роботи з файлом його необхідно закрити.

Розглянемо реалізацію базових операцій при використанні обох підходів.

Бібліотечні функції <stdio.h>

Для відкриття файлу використовуються функція `fopen()`, форматованого виводу – сімейство функцій `printf()` (`fprintf()`), вводу — сімейство функцій `scanf()` (`fscanf()`), закриття файлу — `fclose()`.

Префікс `f` в імені функції означає, що вона призначена для роботи з файлами. При цьому всі основні властивості бібліотечних функцій зберігаються.

Функція `fopen()` має наступний синтаксис:

```
FILE* fopen (const char * ім'я_файлу, const char * режим)
```

з параметрами:

- `ім'я_файлу` — ім'я файлу, наприклад, `C:\\WINDOWS\\system.ini`.
- `режим` — режим доступу до файлу.

У стандарті мови C/C++ визначено наступні режими доступу до файлів.

Табл. 5.1. Режими доступу до файлів

Режим доступу	Опис
`r`	Доступ тільки для читання. Застосовний тільки для існуючого файлу.
`w`	Доступ для запису. Якщо файл вже існує, його вміст стирається. Якщо файл не існує, він створюється.
`a`	Доступ для додавання нової інформації. Якщо файл вже існує, дані додаються в кінець. Якщо файл не існує, то він створюється.
`r+`	Доступ для модифікації. Застосовний тільки для існуючого файлу.
`w+`	Створити пустий файл для читання та запису. Якщо файл вже існує, його вміст стирається.
`a+`	Доступ для читання та додавання. Якщо файл вже існує, дані додаються в кінець. Можна читати дані з будь-якої частини файлу. Якщо файл не існує, то він створюється.

В кожному з режимів доступу є можливість вказати тип файлу `b` для двійкового файлу або `t` для текстового (за замовчуванням).

Функція повертає вказівник на структуру `FILE`, або `0` у разі помилки. Нижче наведено приклад відкриття файлу. У разі виникнення помилок виводиться повідомлення.

```
#include <stdio.h>

int main()
{
  using namespace std;

  // Відкрити існуючий файл для читання
  FILE * my = fopen("myfile.txt", "r");

  if(!my)
     printf("Не можна відкрити файл!");
}
```

Найбільш поширеною функцією для виведення даних у файл є функція `fprintf()` з наступним синтаксисом:

```
int fprintf (FILE * потік, const char * формат, ...)
```

з параметрами:

- `потік` — вказівник на структуру `FILE`, отриманий при відкритті файлу.
- `формат` — форматна стрічка. Довільний рядок, що може містити специфікатори формату наступного вигляду

```
%[флаги][ширина][.точність][довжина]специфікатор
```

Найважливішим є поле `специфікатор`, яке визначає тип та формат даних, що буде записано у файл.

Табл. 5.2

Поле `специфікатор`	Що буде виведено/прочитано	Приклад
`c`	Символ	A
`d` або `i`	Ціле число зі знаком	-234
`f`	Число з плаваючою точкою	34.32
`s`	Рядок С	Hello, World!
`u`	Ціле без знаку	123

Можливі значення інших полів є наступними.

Табл. 5.3

Поле `flags`	Опис
`-`	Вирівнювання вліво. За замовчуванням — вправо
`' '`	Якщо знак не відображається, перед числом вставляється пробіл
`0`	Використати нулі замість пробілів при вирівнюванні вліво

Табл. 5.4

Поле `width`	Опис
`число`	Мінімальна кількість символів при виведенні значення. У випадку, коли значення менше цього числа, результат доповнюється пробілами.
`' '`	Якщо знак не відображається, перед числом вставляється пробіл.

Табл. 5.5

Поле `.precision`	Опис
`.число`	Для `d`, `i`, `u` — мінімальна кількість знаків, що буде виведено. Якщо результат менше цього числа, поле вирівнюється нулями зліва. Для `f` — кількість знаків після точки. Для `s` — максимальна кількість символів на виході.

Табл. 5.6

Поле `length`	Опис
`H`	Аргумент інтерпретується як `short int` або `unsigned short int` (для `i`, `d`, `u`)
`l`	Аргумент інтерпретується як `long int` або `unsigned long int` (для `i`, `d`, `u`)
`L`	Аргумент інтерпретується як `long double` (для `f`)

У якості інших аргументів вказуються значення, що будуть виведені у відповідності до визначеного форматного рядку.

Приклад:

```
fprintf(stdout, "%3.1f", 3.141)
 // вивести число PI з точністю 1 знак після коми,
3.1
```

Функція `fprintf()` повертає кількість символів, що було записано в файл.

Для читання інформації з файлу використовується функція з наступним синтаксисом (табл. 5.7)

```
int fscanf (FILE * stream, const char * format, ...)
```

з параметрами:

- `stream` — вказівник на структуру `FILE`, отриманий при відкритті файлу;
- `format` — форматний рядок. Це довільний рядок, що може містити специфікатори формату виду

```
[=%[*][width][modifiers]type=]
```

Табл. 5.7

`*`	Дані будуть прочитані, але не будуть записані у відповідний аргумент.
`width`	Максимальне число символів, що буде прочитано в поточній операції.
`modifiers`	Те ж саме, що і `length` в `fprintf()`
`type`	Тип змінної, що буде прочитано. Те ж, що і `specifier` у `fprintf()`. Підтримуються `c`, `d`, `f`, `s`, `u`.

Інші аргументи можуть змінюватись в залежності від форматного рядку. Вони визначають послідовність адрес змінних, у які будуть зчитані дані з файлу.

Приклад 5.2

Прочитати з файла `test.in` 2 числа і записати їх суму в файл `test.out`.

```
#include <cstdio>
#include <cstdlib>

int main()
{
  using namespace std;

  const char
  *input = "test.in",
  *output = "test.out";
  int a, b;

  // Відкрити вхідний файл тільки для читання
  FILE * in = fopen(input, "r");
  if(!in)
  {
      perror(input);
      return -1;
  }

  // Прочитати 2 цілих числа
  if(fscanf(in, "%d %d", &a, &b) != 2)
  {
      printf("Не можу прочитати 2 цілих числа з
файлу.\n");
      return -2;
  }
  // Відкрити вихідний файл для запису
  FILE * out = fopen(output, "w");
  if(!out)
  {
      perror(output);
      return -3;

  }

  // Записати суму чисел у вихідний файл
  fprintf(out, "%d", a + b) ;
      return 0;
}
```

Іноді виникає необхідність прочитати або записати у файл складний тип даних, для якого не визначено окремих символів форматування (наприклад, структуру або масив). Для цього використовують бінарні файли та функції

```
size_t fread (void * ptr, size_t size, size_t count,
FILE * stream); // Читання з файлу
size_t fwrite (const void * ptr, size_t size, size_t
count, FILE * stream); // Запис в файл
```

з наступними параметрами:

- `ptr` — вказівник на масив, що буде прочитано з файлу або записано в файл;
- `size` — розмір елементу масиву в байтах;
- `count` — кількість елементів в масиві;
- `stream` — вказівник на структуру `FILE`, отриманий при відкритті файлу.

Функція `fread()` (`fwrite()`) повертає кількість прочитаних (записаних) елементів.

Приклад 5.3

Читання та запис заголовку двійкового файлу.

```
// Заголовок бінарного файлу
struct {
  int version;
  char name[20];
  int data_size;
} header;

FILE* input = fopen("test.hdf", "r+b")
// Прочитати заголовок з бінарного файлу
if(!fread(&header, sizeof header, 1, input))
{
     printf("Не можу прочитати заголовок!");
     exit(-1);
}

printf("%d", header.version);
header.version++;
```

```
// Записати заголовок в бінарний файл
if(!fwrite(&header, sizeof header, 1, input))
{
     printf("Не можу записати заголовок!");
     exit(-2);
}
```

Для переміщення в рамках файлу (переміщення вказівника поточної позиції) використовується функція

```
int fseek (FILE * stream, long int offset, int
origin)
```

з параметрами:

- `stream` — вказівник на структуру `FILE`, отриманий при відкритті файлу;
- `offset` — зсув відносно `origin`;
- `origin` — позиція, відносно якої відраховується `offset`. Цей параметр може приймати наступні значення:
- `SEEK_SET` — початок файлу;
- `SEEK_CUR` — поточна позиція в файлі;
- `SEEK_END` — кінець файлу.

Функція повертає `0`, якщо переміщення було успішним.

Наприклад,

```
fseek(f, 20, SEEK_SET);
// Перемістити вказівник у файлі на 20 байт
```

Для того, щоб отримати поточну позицію у файлі використовують функцію

`long int ftell ( FILE * stream)`, де `stream` — вказівник на структуру `FILE`, отриманий при відкритті файлу.

Наприклад, знайти розмір файлу можна наступним чином.

```
FILE* f = fopen("test.txt", "r");
fseek(f, 0, SEEK_END);
long size = ftell(f);
```

Закрити файл можна за допомогою функції `int fclose (FILE * stream)` з параметром `stream` — вказівник на `FILE`, що закривається.

Приклад:

```
fclose(in);
```

Потоки C++

Більш сучасний підхід для роботи з файлами з'явився у стандарті мови C++. В даному підході використовуються потоки введення/виведення `ifstream` та `ofstream`.

Потоки `ifstream` та `ofstream` є класами (class). Більш докладно класи, їх властивості та основні принципи використання будуть розглядатися в учбовому курсі **Програмування. Об'єктно-орієнтований підхід.**

Важливим є те, що операції та функції, які застосовні до стандартних потоків введення/виведення `cin` та `cout` (ЛР №1), можна використовувати також і при роботі з `ifstream` та `ofstream`.

Відкрити файл можна при створенні потоку:

```
ifstream ( const char * filename, mode);
ofstream ( const char * filename, mode);
```

або за допомогою відповідних їх методів

```
void open ( const char * filename, mode);
void open ( const char * filename, mode);
```

з наступними параметрами:

- `filename` — ім'я файлу, наприклад /etc/passwd;
- `mode` — режим доступу до файлу.

У стандарті C++ визначено наступні режими доступу до файлу:

Табл. 5.8

`app`	Доступ для додавання нової інформації. При виводі інформація додається в кінець файлу.
`ate`	Перемістити вказівник файлу в кінець.
`binary`	Режим доступу до бінарного файлу.
`in`	Доступ для читання.
`out`	Доступ для запису.
`trunc`	Створити пустий файл для читання та запису. Якщо файл вже існує, його вміст стирається.

Режими доступу можуть поєднуватися за допомогою операції 'або' (`|`). Наприклад, режим `ios::binary | ios::in` визначає доступ тільки для читання бінарного файлу.

Відкрити файли за допомогою потоків введення/виведення можна наступним чином

```
// Відкрити файл для читання
ifstream in1("test.in");

// Те ж, з використанням функції open()
ifstream in2;
in2.open("test.in");

// Відкрити файл для запису
ofstream out1("test.out");

// Те ж, з використанням функції open()
ifstream out2;
out2.open("test.out");
```

Потоки `ifstream` та `ofstream` дозволяють використовувати модифікатори (`endl`, `width`, `setf`, `hex`) та методи (`getline()`), що визначені також і для потоків `cin` та `cout` (див. ЛР № 1). Наприклад, записати ціле число в шістнадцятковій системі в файл можна наступним чином:

```
ofstream o("test.txt");
o << hex << 1234;
```

Приклад 5.4

Прочитати 2 числа з вхідного файлу та записати в вихідний файл їх суму.

```
#include <iostream>
#include <fstream>

int main()
{
    using namespace std;

    ifstream in("test.in");
    if(!in)
    {
        cerr << "Cannot open input file!\n";
        return -1;
    }
```

```
    int a, b;
    in >> a >> b;

    ofstream out("test.out");
    if(!out)
    {
        cerr << "Cannot open output file!\n";
        return -2;
    }

    out << a + b;

    return 0;
}
```

Закрити файл можна за допомогою методу потоку `void close()`. Наприклад,

```
ifstream i("test.txt");
...
i.close();
```

Приклад 5.5

Виконання операцій над даними, зчитаними із файлу.

У наведеному нижче прикладі з файлу зчитується матриця, з якою далі виконуються прості операції.

```
#include <iostream>
#include <fstream>

int main()
{
    using namespace std;

   // Відкрити файл тільки для читання
    ifstream in("tst4.in");

    if(!in)
    {
        cerr << "Не можу відкрити вхідний файл!";
        return -1;
    }
```

```
    int n, m;

    in >> n >> m;

  // Створити матрицю
    int** mat = new int* [n];
    for(int i = 0; i < n; ++i)
      mat[i] = new int[m];

  // Прочитати матрицю з файлу
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < m; ++j)
          in >> mat[i][j];

    double sum = 0;

  // Знайти суму всіх елементів матриці
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < m; ++j)
          sum += mat[i][j];

  // Відкрити файл для запису
    ofstream out("tst4.out");

    if(!out)
    {
        cerr << "Не можу відкрити вихідний файл!";
        return -2;
    }


  // Записати матрицю з коефіцієнтами розділеними на
  // суму її елементів
    for(int i = 0; i < n; ++i, out << endl)
      for(int j = 0; j < m; ++j)
          out << '\t' << mat[i][j] / sum;
}
```

5.2. Варіанти завдань

Обов'язкові завдання

1. В файлі `test.in` записно цілі числа. Знайти їх суму. Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `cstdio`.
2. В файлі `test.in` записно цілі числа. Знайти їх суму. Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `fstream`.
3. В файлі `test.in` записано матрицю *N*x*N*. Знайдіть її детермінант. Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `cstdio`.
4. В файлі `test.in` записано матрицю *N*x*N*. Знайдіть її детермінант. Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `fstream`.
5. В файлі `test.in` записано текст англійською мовою. Запишіть в файл test.out всі рядки з файлу `test.in`, в яких зустрічається слово "*Hello*". Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `cstdio`.
6. В файлі `test.in` записано текст англійською мовою. Запишіть в файл test.out всі рядки з файлу `test.in`, в яких зустрічається слово "*Hello*". Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `fstream`.
7. В файлі `test.in` записано текст англійською мовою. Замініть всі входження слова "*Hello*" на слово "*World*". Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `cstdio`.
8. В файлі `test.in` записано текст англійською мовою. Замініть всі входження слова "*Hello*" на слово "*World*". Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `fstream`.
9. В файлі `test.in` записано текст англійською мовою. Змініть регістр алфавітних символів. (Приклад: "*Hello*, *World*!" стане "*hELLO*, *wORLD*!"). Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `cstdio`.
10. В файлі `test.in` записано текст англійською мовою. Змініть регістр алфавітних символів. (Приклад: "*Hello*, *World*!" стане "*hELLO*, *wORLD*!"). Результат запишіть в `test.out`. Для роботи з файлами використовуйте функції `fstream`.

Завдання середнього рівня

1. Створіть програму, що буде видаляти з лістингу програми на мові C++ (файл *.cpp) коментарі виду /* коментар */. Для роботи з файлами використовуйте функції `cstdio`.
2. Створіть програму, що буде видаляти з лістингу програми на мові C++ (файл *.cpp) коментарі виду /* коментар */. Для роботи з файлами використовуйте функції `fstream`.
3. Створіть програму, що буде видаляти з лістингу програми на мові C++ (файл *.cpp) коментарі виду // коментар. Для роботи з файлами використовуйте функції `cstdio`.
4. Створіть програму, що буде видаляти з лістингу програми на мові C++ (файл *.cpp) коментарі виду // коментар. Для роботи з файлами використовуйте функції `fstream`.
5. Створіть програму, що буде видаляти в текстовому файлі символи-роздільники (пробіл, символ табуляції) в кінці строк. Для роботи з файлами використовуйте функції `cstdio`.
6. Створіть програму, що буде видаляти в текстовому файлі символи-роздільники (пробіл, символ табуляції) в кінці строк. Для роботи з файлами використовуйте функції `fstream`.
7. Створіть програму, що буде підраховувати частоти монограм (байтів) в бінарному файлі. Який байт найчастіше зустрічається в текстовому файлі (*.txt)? Виконуваному файлі (*.exe)? Для роботи з файлами використовуйте функції `cstdio`.
8. Створіть програму, що буде підраховувати частоти монограм (байтів) в бінарному файлі. Який байт найчастіше зустрічається в текстовому файлі (*.txt)? Виконуваному файлі (*.exe)? Для роботи з файлами використовуйте функції `fstream`.
9. Створіть програму, що буде підраховувати ентропію за Шенноном бінарного файлу ($H = - \sum_{i=0}^{255} f_i \log f_i$, де f_i -- частота входження байту i). Знайдіть ентропію документу Word (*.doc) та архіву (*.zip). Для роботи з файлами використовуйте функції `cstdio`.
10. Створіть програму, що буде підраховувати ентропію за Шенноном бінарного файлу ($H = - \sum_{i=0}^{255} f_i \log f_i$, де f_i -- частота входження байту i). Знайдіть ентропію документу Word (*.doc) та архіву (*.zip). Для роботи з файлами використовуйте функції `fstream`.
11. Створіть програму, що буде в текстовому файлі переводити символи табуляції в пробіли (символ табуляції переводить курсор вперед на позицію кратну 8). Для роботи з файлами використовуйте функції `cstdio`.

12. Створіть програму, що буде в текстовому файлі переводити символи табуляції в пробіли (символ табуляції переводить курсор вперед на позицію кратну 8). Для роботи з файлами використовуйте функції `fstream`.
13. Створіть програму, що буде в текстовому файлі переводити пробіли в символи табуляції (символ табуляції переводить курсор вперед на позицію кратну 8). Для роботи з файлами використовуйте функції `cstdio`.
14. Створіть програму, що буде в текстовому файлі переводити пробіли в символи табуляції (символ табуляції переводить курсор вперед на позицію кратну 8). Для роботи з файлами використовуйте функції `fstream`.
15. Створіть программу пошуку входження строки в бінарному файлі. Чи входить строка 'This program' у виконуваний файл Вашої програми (*.exe)? Для роботи з файлами використовуйте функції `cstdio`.
16. Створіть программу пошуку входження строки в бінарному файлі. Чи входить строка '*This program*' у виконуваний файл Вашої програми (*.exe)? Для роботи з файлами використовуйте функції `fstream`.
17. Створіть програму, що буде знаходити всі текстові строки довжиною більше 5 символів в бінарному файлі. Для роботи з файлами використовуйте функції `cstdio`.
18. Створіть програму, що буде знаходити всі текстові строки довжиною більше 5 символів в бінарному файлі. Для роботи з файлами використовуйте функції `fstream`.
19. Створіть програму, що буде виводити шістнадцятковий дамп бінарного файлу (замість кожного байту вхідного файлу виводиться значення в шістнадцятковій системі). Для роботи з файлами використовуйте функції `cstdio`.
20. Створіть програму, що буде виводити шістнадцятковий дамп бінарного файлу (замість кожного байту вхідного файлу виводиться значення в шістнадцятковій системі). Для роботи з файлами використовуйте функції `fstream`.
21. Створіть програму, що буде виводити вісімковий дамп бінарного файлу (замість кожного байту вхідного файлу виводиться значення в вісімковій системі). Для роботи з файлами використовуйте функції `cstdio`.
22. Створіть програму, що буде виводити вісімковий дамп бінарного файлу (замість кожного байту вхідного файлу виводиться значення в вісімковій системі). Для роботи з файлами використовуйте функції `fstream`.
23. Створіть програму, що буде виводити десятковий дамп бінарного файлу (замість кожного байту вхідного файлу виводиться значення в десятковій системі). Для роботи з файлами використовуйте функції `cstdio`.

24. Створіть програму, що буде виводити десятковий дамп бінарного файлу (замість кожного байту вхідного файлу виводиться значення в десятковій системі). Для роботи з файлами використовуйте функції `fstream`.

Завдання підвищеної складності

1. Створіть програму, що підраховує частоти біграмм у бінарному файлі. Що таке біграмма можна прочитати в додатковій літературі [А.П.Алферов, А.Ю.Зубов, А.С.Кузьмин, А.В.Черемушкин "Основы криптографии"].
2. Створіть програму, що підраховує частоти входження слів в текстовому файлі.
3. Створіть програму, що підраховує частоти N-грамм ($N > 2$) у бінарному файлі. Більш детально про статистичний аналіз в криптографії, N-грами можна дізнатися в додатковій літературі [А.П.Алферов, А.Ю.Зубов, А.С.Кузьмин, А.В.Черемушкин "Основы криптографии"].

5.3. Контрольні запитання

1. Які засоби роботи з файлами є в мові C++?
2. Які існують методи відкриття файлів?

Комп’ютерний практикум №6
Робота з рядками

Мета роботи: отримати навички роботи з рядками засобами бібліотеки STL C++ та C.

6.1. Теоретичні відомості

Простір імен

Простір імен (**namespace**) — це елемент мови C++, призначений для розв'язання конфліктів імен у програмах, що складаються з декількох файлів. Оголошення простору імен здійснюється наступним чином.

```
namespace [ідентифікатор]
{
  // опис робочої області
}
```

Звернутися до ідентифікатору, що належить деякому простору імен, можна за допомогою оператора розширення контексту `::`. Якщо виникає необхідність часто звертатися до ідентифікаторів з певного простору імен, зручно використовувати конструкцію `using namespace`. Це дозволяє уникнути необхідності використання простору імен при кожному зверненні, наприклад:

```
using namespace [ідентификатор]
```

Приклад 6.1. Оголошення та використання декількох просторів імен

```
#include <iostream>
using namespace std;

namespace spaceA
{
  int MyVal=10;
}
```

```
namespace spaceB
{
 int MyVal=20;
}

namespace spaceC
{
 int MyVal=30;
}

void Test()
{
 using namespace spaceB;
 cout << MyVal << " " << "spaceB" << endl;
}

int main()
{
 using namespace spaceA;
 cout << MyVal << " " << "spaceA" << endl;
 Test();
 cout << spaceC::MyVal << " " << "spaceC" << endl;
 return 0;

}
```

Рядки в C++

В бібліотеці STL C++ є дуже корисна бібліотека роботи з рядками. Вона досить ефективна і дозволяє легко вирішувати наступні задачі:
1. Створювати, присвоювати, копіювати і видаляти рядки.
2. Виконувати перетворення типів символьних змінних.
3. Порівнювати рядки.
4. Поєднувати рядки.
5. Визначати довжину рядка.
6. Знаходити і заміщати потрібний фрагмент у рядку.

Для використання рядків, необхідно підключити відповідний заголовний файл `<string>`. Після цього, операція створення нового рядка виявиться настільки ж простою, як і створення змінної будь-якого базового типу.

```
string hi("hello");
// створення й ініціалізація нового рядка
string lo="greetings"; // ще одна ініціалізація
string es=""; // порожній рядок
```

Рядковим змінним можна присвоювати значення, як і змінним будь-яких інших типів:

```
string name("Fred");
name = "Flintstone"; // зміна імені
```

При цьому всі операції розподілу пам'яті будуть виконані коректно. Операції введення та виведення (<< і >>) перевантажені і для рядків, тому виконати введення або виведення рядка дуже легко.

Приклад 6.2. Використання потокових операцій введення-виведення для рядків

```
#include <iostream>
//заголовний файл з бібліотеки STL
#include <string>
using namespace std;
int main()
{
  string s;
  while (cin >> s)
  //зчитування кожного слова зі стандартного
  //потоку cin
  cout << s << endl;
  //виведення кожного слова в окремому рядку
  return 0;
}
```

Операція + перевантажена для рядків і означає конкатенацію. Операції порівняння (==, > і т.д.) теж перевантажені для рядків.

```
…
string hi("hello");
string lo="greetings";
string r = hi + ' ' + "world";
//конкатенація трьох рядків
r += '!'; // так теж можна! (r = "hello world!")
r = string(3, '!'); // r = "!!!";
if (hi == "hello")
if (lo > "great")
if (lo < hi)
```

…

У бібліотеці міститься ще декілька корисних функцій, використання яких демонструється на наступному прикладі:

```
cout << hi.find("ll");
//повертає 2 (позицію самого лівого входження "ll")
cout << hi.find('l');
//працює завдяки автоматичному приведенню типів
cout << hi.rfind('l');
//виконує пошук з кінця (повертає 3)
cout << lo.find("g");
//повертає 0 (положення самого лівого входження
//'g')
cout << lo.find("g", 5);
//повертає 7 (саме ліве входження 'g' після 5).
cout << hi.find("x") // повертає string::npos
string s("Testing!");
cout << s.substr(2, 5); // "sting"
cout << s[3]; // 't'
s.replace(2,2,"eth"); // s == "Teething!"
s.erase(1,4); // s == "Ting!"
s[1] = 'a'; // s == "Tang!"
s.insert(1,"w"); // s == "Twang!"
```

Приклад 6.3. Основні операції над рядками

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string s0 = "abcde";
  string s1 = " fg";

  // Конкатенація рядків.
  string s = s0 + s1;
  cout << s << "\n";

  // Отримання символа у потрібній позиції
  char ch0 = s0.at(1);
  cout << ch0 << "\n";
```

```
  char ch1 = s0[3];
  cout << ch1 << "\n";

  // Рядок порожній ?
  if (s0.empty())
  {
    cout << "String is empty" << "\n";
  }
  else
  {
    cout << "String isn't  empty" << "\n";
  }

  // Встановлюємо значення й порівнюємо 2 рядка.
  s1 = s0;
  if(s1 == s0)
  {
    cout << "Strings are equal" << "\n";
  }
  else
  {
    cout << "Strings are not equal" << "\n";
  }

  // Читання введеного з клавіатури рядка.
  getline(cin, s1);
  cout << s1<<endl;

  // Одержання довжини рядка.
  cout << s1.length();
  return 0;
}
```

Приклад 6.4. Створення та ініціалізація рядків

```
#include <string>
using namespace std;
int main()
{
  string hi("hello");
  // створення й ініціалізація нового рядка
```

```
    string lo="greetings"; // ще одна ініціалізація
    string g(lo); // третя ініціалізація
    string es=""; // порожній рядок
    string s; // ініціалізація порожнього рядка
    return 0;
}
```

6.2. Порядок виконання роботи

1. Проаналізувати умову задачі.
2. Розробити алгоритми та створити програми розв'язання задачі згідно з номером варіанту.
3. Результати роботи оформити протоколом.

6.3. Варіанти завдань

Обов'язкові завдання

1. Знайти кількість входжень заданого символу в заданий рядок.
2. Отримати рядок, що являє собою конкатенацію двох рядків.
3. Визначити довжину рядка.
4. Порівняти два введених рядки.
5. Замінити всі входження символу '*а*' в рядку на символ '*б*'.
6. Замінити перший та четвертий символ рядка на букву '*а*'.

Завдання підвищеної складності

1. Ввести два рядки, порівняти їхню довжину, перший і останній символи кожного рядка, а також вивести ці рядки із заголовної літери.
2. Написати програму, що перевіряє, чи є частиною даного слова слово "*сос*". Якщо відповідь негативна, то додати до введеного слова слово "*не*" у початок і кінець. Якщо відповідь "*так*", то перевірити, чи не є воно словом «*сосна*».
3. Написати програму, яка задає користувачеві запитання, що вимагає однозначної відповіді. Перевірити його правильність. Дати користувачеві кілька підказок і спроб. Якщо він вгадав, то запитати його ім'я, і вивести на екран поздоровлення, що є конкатенацією декількох рядків, двічі вживши його ім'я.

4. Запропонувати користувачеві ввести дату в форматі *ДД-ММ-РР*. День і місяць можуть бути зазначені одноцифровими числами, тобто 1-5-94, а не 01-05-94. Виділити числа, які позначають день, місяць та рік, і вивести кожне число з відповідним пояснювальним написом.

5. Написати програму, яка випадковим чином загадує літеру латинського (російського) алфавіту. Користувачеві пропонується відгадати загадану літеру, допомагаючи йому в такий спосіб. Якщо в черговій спробі користувачем введена літера, що стоїть ближче до загаданої, ніж попередня, то виводиться користувачеві повідомлення "*Гарячіше*!", а якщо далі - "*Холодніше*!".

6. З'ясувати, яка з літер (перша чи остання) зустрічається в заданому слові частіше.

7. Написати програму шифрування та дешифрування текстового повідомлення. Можна використати такий спосіб шифрування. Шифрувальник задає ключ шифрування - ціле число, що визначає величину зсуву літер російського алфавіту, наприклад ключ =3, тоді в тексті літера "*а*" замінюється на "*г*" і т.д.

8. У тексті, що складається з латинських літер і закінчується крапкою, підрахувати кількість голосних літер.

9. Написати програму, що з'ясовує, чи пишеться дане слово однаково зліва направо та справа наліво, наприклад: *ПОТОП*, *КОК*).

10. Викреслити зі слова *X* ті літери, які зустрічаються в слові *Z*.

11. Написати програму, що вводить рядок і виводить його, скорочуючи щоразу на 1 символ доти, доки в рядку не залишиться 1 символ.

12. Написати програму, що підраховує, скільки разів у даному слові *X* зустрічається дане слово *У*. Якщо слово *У* довше, ніж *X*, то результат повинен дорівнювати нулю.

13. Дано два тексти *А* та *Б*. Перевірити, чи можна з літер, що входять в *А*, скласти *Б*. (Літери можна переставляти, але кожну літеру можна використати не більше одного разу).

14. Записати рядок *X* у зворотному порядку в рядок *У*. Порахувати, скільки однакових літер знаходяться на однакових місцях у цих рядках.

15. Задано прізвище, ім'я та по батькові студента, розділені пробілом. Надрукувати його прізвище та ініціали.

16. Написати програму, яка рахує кількість цифр у введеному рядку символів.

17. Написати програму, яка рахує кількість літер у введеному рядку символів.

18. Написати програму заміни в рядку всіх літер "*а*" на літери "*б*" і навпаки (при такій заміні, наприклад, зі слова "*баба*" має вийти слово "*абаб*"). Вивести отриманий рядок на екран.

19. Написати програму, що потроює кожну літеру в заданому тексті (при цьому, наприклад, зі слова "*сад*" повинне вийти слово "*сссаааддд*"). Вивести отриманий рядок на екран.

20. Викреслити *i*-у літеру слова.

21. Написати програму, що викреслює з даного тексту будь-яку літеру. Вивести отриманий рядок на екран. Якщо такого символу немає, то вивести відповідне повідомлення.

22. Видалити всі символи в рядку, які стоять після "*". Вивести отриманий рядок на екран. Якщо такого символу немає, то вивести відповідне повідомлення.

23. Вставити в рядок слово за умовою:

а) в кінець рядка;

б) на початок рядка;

в) після першого слова.

Вивести отриманий рядок.

24. Всі слова, у яких літера "*а*" зустрічається більше 2х разів, видалити з тексту. Вивести отриманий рядок на екран. Якщо такого символу немає, то вивести відповідне повідомлення.

25. З рядка видалити середню літеру, якщо довжина рядка непарне число, інакше - видалити дві середні літери. Вивести отриманий рядок на екран.

26. Дано рядок тексту. У даному рядку поміняти місцями кожні два слова із чотирьох перших. Якщо кількість слів менша заданої, то вивести про це повідомлення.

27. Написати програму знаходження слово максимальної довжини у заданому тексті.

28. Написати (у порядку появи в тексті) всі слова, довжина яких попадає в інтервал [*X*, *Y*]. Тут *X* і *Y* цілі числа, що задаються користувачем.

29. У даному реченні знайти кількість слів, що містять подвоєну приголосну (букви латинські). Слова в реченні розділяються пробілами, наприкінці речення - крапка.

30. Написати програму, яка запитує у користувача рядок і символ, виводить на екран повідомлення, чи є серед символів рядка заданий користувачем символ. Якщо - ні, то додає в рядок цей символ на вибір: у початок або в кінець рядка.

31. Написати програму, яка пропонує користувачеві ввести число в інтервалі від 1 до 5 включно. Програма повинна дозволяти користувачеві

вводити будь-яку послідовність символів. Організувати перевірку введення, і якщо введення довше одного символу, або нецифрове, або не потрапляє в зазначений інтервал, тоді вивести повідомлення про помилку і запропонувати користувачеві повторити спробу.

32. З'ясувати, скільки разів зустрічається кожна літера алфавіту в запропонованому тексті.

33. У рядку будь-яку кількість підряд стоячих пробілів замінити одним пробілом.

34. Обчислити довжину найкоротшого слова в реченні із трьох слів, розділених пробілами.

35. Написати (у порядку появи в тексті) всі слова, довжина яких попадає в інтервал [*X*, *Y*]. Тут *X* і *Y* цілі числа, що вказують, відповідно, найбільшу й найменшу довжину.

36. Складіть програму, що викреслює кожну третю літеру заданого слова *X* у заданому реченні.

37. Написати програму, що змінює порядок слів у рядку за Вашим алгоритмом.

38. Скільки літер "*в*" у слові стоїть на парних місцях?

6.4. Контрольні запитання

1. Для чого призначена бібліотека STL C++?
2. Як у мові C++ можна працювати з символьними рядками?
3. Для чого використовується поняття простору імен?
4. Які функції роботи з рядками ви знає

Комп'ютерний практикум №7
Обробка виключень

Мета роботи: отримати навички роботи з механізмом обробки виняткових ситуацій мови C++ та ознайомитись з принципами використання командного рядка для запуску програм.

7.1. Теоретичні відомості

Використання виключень

Механізм обробки помилок (або виняткових ситуацій, виключень — *exception handling*) вбудовано в мову C++. Цей механізм дозволяє коректно обробляти помилки, що виникають в процесі роботи програми. За його допомогою при виникненні помилки може бути автоматично викликана процедура її обробки. Принципова перевага такого підходу полягає в тому, що він дозволяє автоматично, в залежності від ситуації, запускати одну з багатьох функцій обробки помилок, які попередньо «вручну» вбудовуються в основну програму. Належним чином запрограмована обробка виняткових ситуацій допомагає створювати дійсно відмовостійкі програми.

Обробка виняткових ситуацій в мові C++ організується за допомогою трьох ключових слів: `try`, `catch` та `throw`. Фрагменти програми, під час виконання яких потрібно забезпечити обробку виняткових ситуацій, розташовуються в блоці `try`. Якщо виняткова ситуація виникає всередині

блоку `try`, вона «збуджується» (ключове слово `throw`), перехоплюється (ключове слово `catch`) та обробляється.

Функції, що викликаються в блоці `try`, також можуть збуджувати виняткову ситуацію. Будь-яка виняткова ситуація повинна перехоплюватися інструкцією `catch`, що розташовується безпосередньо за відповідним блоком `try`. Далі представлена основна форма інструкцій `try` та `catch`:

```
try {
// блок генерації виняткової ситуації
}
catch (type1 arg) {
// блок перехоплення виняткової ситуації
}
catch (type2 arg) {
// блок перехоплення виняткової ситуації
}
…
catch (type arg) {
// блок перехоплення виняткової ситуації
}
```

У блоках `try` повинні бути розташовані ті частини програми, при виконанні яких необхідно відслідковувати можливі помилки. Це можуть бути як кілька інструкцій всередині однієї функції, так і всі інструкції функції `main()`. Після генерації виняткової ситуації вона перехоплюється та обробляється у відповідному блоці-оброблювачі `catch`. З блоком `try` може бути зв'язано декілька інструкцій `catch`. Те, яка саме інструкція `catch` буде використана, залежить від типу виняткової ситуації. Іншими словами, якщо тип даних, зазначений в інструкції `catch`, відповідає типу

(значенню) згенерованої виняткової ситуації, виконується дана інструкція `catch`. При цьому решта інструкцій блоку `try` буде проігноровано (тобто відразу після того, як якась інструкція в блоці `try` зумовила появу виняткової ситуації, керування передається відповідній інструкції `catch`, минаючи решту інструкцій блоку `try`). Якщо виняткова ситуація перехоплена, аргумент `arg` буде містити відповідне значення. Якщо доступ до значення виняткової ситуації не потрібен, в інструкції `catch` можна вказати лише її тип `type`, а аргумент `arg` вказувати не обов'язково. Можна перехоплювати значення будь-яких типів даних, у тому числі структур та класів.

Ось основна форма інструкції `throw`:

```
throw <значення, що ідентифікує виключення>;
```

Інструкція `throw` повинна виконуватися або усередині блоку `try`, або в будь-якій функції, що викликається у цьому блоці.

При генерації виняткової ситуації, для якої немає відповідної інструкції `catch`, може відбутися аварійне завершення програми. Якщо компілятор відповідає стандарту C++, то генерація необроблюваної виняткової ситуації приведе до виклику стандартних бібліотечних функцій `unexpected()` та `terminate()`. За замовчуванням для завершення програми функція `terminate()` викликає функцію `abort()`, однак при необхідності можна задати власну процедуру завершення програми.

Стандартні дії, що виконуються функціями `terminate()`, `abort()` та `unexpected()`, більш докладно розглядаються у літературі та довідковій системі середовища розробки. Крім цього слід зазначити, що поведінка цих функцій залежить від версії компілятора, що використовується.

Змінити логіку роботи функції `terminate()` можна за допомогою функції `set_terminate()`, а функції `unexpected()` — за допомогою функції `set__unexpected()`.

Блок `catch` може обробити більше одного джерела виключень.

Для уточнення опису функції до її прототипу та визначення можна додати визначення виключень, які вона може генерувати, що складається з ключового слова `throw` і наступного за ним в дужках списку типів виключень.

```
double mean(double a, double b) throw(char *);
```

При цьому, по-перше, компіляторові передається інформація про те, якого роду виключення можуть бути згенеровані функцією. Якщо функція згенерує інший вид виключення, буде викликана функція `abort()`. По-друге, використання специфікації виключень дозволяє будь-якому користувачеві зробити висновок щодо їх переліку і, таким чином, використати цю інформацію у своїй програмі. Наприклад, наступний прототип описує функцію, що може генерувати як виключення `char *`, так і виключення `double`.

```
double multipleErrorsFunction(double z) throw(char
*, double);
```

Слід зазначити, що інформація про виключення повинна міститися як у прототипі, так і у заголовку функції. При цьому порожні дужки у специфікації виключень вказують на те, що функція не генерує виключень.

```
double simple(double z) throw ();
 // не генерує виключень
```

Якщо блок `try` є вкладеним, можна забезпечити, що пов'язані з ним оброблювачі виключень будуть передавати керування оброблювачам

виключень зовнішнього блоку `try`. Щоб перехопити будь-яке виключення, замість вказівки типу виключень слід використовувати три крапки `(...)`:

```
catch(...)
{
    // оператори
}
```

Щоб передати керування зовнішньому блоку `try`, слід використовувати ключове слово `throw` без параметру.

```
catch (char* s)
{
      cout << "Exception caught in inner loop." <<;
      throw;
//перенаправлення до зовнішнього блоку try
}
```

У наведеному фрагменті програмного коду на екран виводиться повідомлення, після чого керування передається зовнішньому блоку `try`, де знову буде виконано пошук оброблювача виключень, що відповідає згенерованому виключенню.

Існує декілька способів включення одного блоку `try` в інший. Перший спосіб полягає у вкладенні одного блоку в інший. Другий спосіб полягає в тому, що один блок `try` містить виклик функції, що активізує функцію з іншим блоком `try`. У першому випадку керування буде передано зовнішньому блоку `try`. У другому випадку для пошуку наступного блоку `try` буде виконано процедуру розгортання стека.

Розгортання стека

Припустимо, що блок `try` не містить прямого виклику функції, що генерує виключення, але викликає функцію, що у свою чергу викликає функцію, що генерує виключення. Виконання програми передається від функції, у якій генерується виключення, функції, що містить блок `try` та відповідні оброблювачі виключень `catch`. Для цього виконується процедура розгортання стека.

В мові C++ виклики та повернення функцій обробляються наступним чином. При виклику функції інформація розміщається в стеку. Зокрема, у стеку програма розміщає адресу інструкції, що викликала функцію (адресу повернення). При завершенні виконання функції, що була викликана, програма використовує зазначену адресу як адресу, з якого продовжується виконання програми. При виклику функції всі її аргументи також записуються в стек, де вони одержують статус змінних з автоматичним класом пам'яті. Якщо у функції, що викликається, створюються нові змінні з автоматичним класом пам'яті, вони, у свою чергу, також додаються в стек. Якщо у даній функції викликається інша функція, то її інформація також додається в стек і т.д. Коли завершується виконання функції, керування передається за адресою, що була збережена при виклику функції, при цьому вершина стека звільняється. Таким чином, після виконання функції відбувається повернення до тієї функції, що її викликала, і т.д., причому при завершенні роботи кожної з таких функцій відбувається звільнення змінних з автоматичним класом пам'яті. Якщо змінні з автоматичним класом пам'яті є об'єктами класу, то викликається деструктор класу (якщо такий є).

Більш докладно робота з класами та об'єктами розглядається у курсі **Об'єктно-орієнтоване програмування**.

Тепер припустимо, що виконання функції переривається не в результаті повернення керування функції, що її викликала, а в результаті виконання блоку обробки виключення. Тоді програма так само, як у випадку нормального повернення, видаляє значення зі стека. Але замість того, щоб зупинитися на першій адресі повернення в стеку, програма буде продовжувати звільняти стек, поки не досягне адреси повернення, що перебуває у відповідному блоці `try`. Після цього керування передається оброблювачам виключень наприкінці цього блоку, а не першому операторові, розташованому зразу за оператором виклику функції. Цей процес називається розгортанням стека. Однією з важливих особливостей механізму генерації виключень є те, що як і при поверненні функцій викликаються деструктори класів для будь-яких автоматичних об'єктів, розташованих в стеку. При цьому при поверненні керування з функції обробляються лише об'єкти, розміщені в стеку цією функцією, у той час як механізм обробки виключень обробляє об'єкти, що були розміщені в стеку при виконанні цілого ланцюга викликів функцій між блоком `try` та місцем генерації виключень. При відсутності властивості розгортання стека при генерації виключень залишилися б невикликаними деструктори для автоматичних об'єктів класів, розміщених у стеку при реалізації проміжних викликів всіх відповідних функцій.

Робота з параметрами командного рядка

Для забезпечення передачі даних у програму на мові C++ через командний рядок операційної системи можна скористатися двома параметрами функції `main()`:

```
void main(int argc, char *argv[])
```

При цьому у першому параметрі `argc` буде розміщено кількість елементів (параметрів) масиву вказівників `argv` на символьні рядки. При цьому кожний символьний рядок відповідає окремому параметру командного рядка. Слід зазначити, що всі значення, що передаються у програму через командний рядок, є строковими. Тому при передачі значень інших типів у програмі необхідно попередньо виконати їх перетворення у потрібний тип даних. У наступному прикладі параметр `argc` використовується для виводу кількості аргументів, переданих через командний рядок операційної системи.

Приклад 7.1. Використання аргументів, переданих через командний рядок

```
#include <iostream>
using namespace std;

void main(int argc, char *argv[])
{
   cout << "Кількість аргументів командного рядка";
   cout << " дорівнює " << argc << endl;
}
```

Якщо ім'ям файлу з програмним кодом є `test_params.cpp`, то після успішної компіляції створену програму можна запустити з командного рядку та передати необхідні дані наступним чином.

```
c:\test_params 1 2 "Third string parameter"
```

Аргументи функції `main()`, які згруповано всередині подвійних лапок, будуть інтерпретуватись як один аргумент.

Параметр `argv[0]` завжди містить ім'я виклику програми, так що значенням змінної `argc` як мінімум є 1.

Як вже зазначалось, другий параметр функції `main()` з ім'ям `argv` є масивом вказівників на символьні рядки з окремими частинами командного рядка (переданими параметрами). У наступному прикладі оператор `for` використовується для виводу елементів масиву `argv`. (Програма виводить кожний елемент, поки значення змінної циклу не стане більше, ніж `argc`).

Приклад 7.2. Використання аргументів, переданих через командний рядок

```
#include <iostream>
using namespace std;

void main(int argc, char *argv[])
{
   int i;
   for (i = 0; i < argc; i++)
        cout << "argv[" << i      << "] містить ";
   cout << argv[i] << endl;
}
```

У мові C++ для помітки закінчення масиву `argv` використовується символ `NULL`. У наступному прикладі за допомогою цикла `for` виконується прохід по елементах масиву `argv`, поки його поточний елемент не буде містити значення `NULL`.

Приклад 7.3. Використання аргументів, переданих через командний рядок

```
#include <iostream>
using namespace std;

void main(int argc, char *argv[])
{
   int i;
   for (i = 0; argv[i] != NULL; i++)
   {
      cout << "argv[" << i << "] містить ";
      cout << argv[i] << endl;
   }
```

Оскільки параметри командного рядка програми на мові C++ передаються як вказівники на рядки, найчастіше їх потрібно перетворювати в необхідний тип. Для цього можна скористатися одною з функцій родини `atof()`, `atoi()`, `atol()`. Повний перелік та опис цих функцій можна знайти в заголовних файлах `math.h` та `stdlib.h`.

Ці функції дозволяють перетворити символьний рядок у значення з плаваючою крапкою подвійної точності (функція `atof()`), у ціле значення (функція `atoi()`) або в довге ціле значення (функція `atol()`). Функції припиняють зчитування вихідного рядка після появи першого символу, що не може бути інтерпретований як частина числа (наприклад, символу `NULL`, що завершує рядок). Функція `atof()` припускає, що її параметр-рядок може мати степеневе представлення `[whitespace][sign][digits][{d|D|E}[sign]digits]`. Функції `atoi()` та `atol()` не обробляють десяткові крапки або показники ступеня. Кожна з функцій повертає значення типу `double`, `int` або `long`, отримане в результаті інтерпретації вхідних символів як чисел. Значення, що повертається, дорівнює `0` (`0L` для функції `atol()`), якщо вхідний

параметр не може бути перетворений у значення даного типу. Значення, що повертається, у випадку переповнення є невизначеним.

Зворотне перетворення з числового представлення у строкове дозволяють виконати функції `ecvt()`, `fcvt()`, `gcvt()`.

У наступних прикладах ілюструється перетворення строкових представлень чисел в числові значення з використанням функцій `atof()`, `atoi()`, `atol()`.

Приклад 7.4. Використання функцій `atof()`, `atoi()`, `atol()`

```
#include <stdlib.h>

main( )
{
   char *s;
   double x;
   int i;
   long l;

   s=" -2309.12E-15";
   x=atof(s);
   cout << "Число з плаваючою крапкою: "<< x;
   cout << endl;

   s="7.8912654773d210";
   x=atof(s);
   cout << "Число з плаваючою крапкою: ";
   cout << x << endl;

   s="-9885";
```

```
    i=atoi(s);
    cout << "Ціле"<< i << endl;

    s="98854 dollars";
    l=atol(s);
    cout << "Довге ціле:  " << l << endl;
}
```

Більш докладно теоретичні питання, пов'язані з даним комп'ютерним практикумом, викладено у [3].

7.2. Приклад

У наступному прикладі показано, як в мові C++ функціонує система обробки виняткових ситуацій.

Приклад 7.5. Обробка виняткових ситуацій

```
#include <iostream>
#include<conio.h>
using namespace std;

int main()
{
   int a;
   char* b = "test string for exception generation";
   cout << "Begining..." << endl;

   //cout << b;
  try {
     // Блок try
```

```
        cin >> a;
        cout << "Inside try block" << endl;

        if(!a) throw 10; // Генерація виключення 10
        if(a == -1) throw 10.;
        // Генерація виключення 10.
        else throw b;
// Генерація виключення зі строковим ідентифікатором

        cout << "This statement can't to be executed";
  }
    catch (int i)
    {           // Оброблювач 1
        cout << "Error with number: ";
        cout << i << endl;
    }
    catch (char* b)
    {           // Оброблювач 2
        cout << "String error: ";
        cout << b << endl;
    }
    catch(...)
    {   // Узагальнений оброблювач
        cout << "Unknown error 10.!" << endl;
        cout << "Program finished ...";

    }
  getch();
  return 0;
}
```

7.3. Порядок виконання роботи

1. Проаналізувати умову задачі.
2. Створити програму розв'язання задачі згідно з номером варіанту.
3. Результати роботи оформити протоколом.

7.4. Варіанти завдань

Обов'язкові завдання

Реалізувати варіанти завдань з комп'ютерного практикуму № 1 з урахуванням наступних додаткових вимог.

1. Математичний вираз повинен обчислюватись у окремій користувацькій функції.
2. Коректність вхідних даних повинна перевірятись за допомогою механізму обробки виключень мови C++.
3. При виконанні завдання забезпечити дворівневу перевірку двома способами:

- за допомогою вкладених блоків `try`;
- шляхом перехвату виключень у основній програмі та у функції, що викликається.

Завдання підвищеної складності

1. Для програми, реалізованої відповідно до пункту 7.4, забезпечити введення вхідних параметрів через командний рядок.

2. За допомогою механізму обробки стандартних виключень та макросу `assert` забезпечити перевірку коректності введених даних.
3. Забезпечити перетворення параметрів командного рядку у необхідні типи даних.

7.5. Контрольні запитання

1. Що таке виключення та виключна ситуація? Як їх обробку можна реалізувати у мові C++?
2. Що таке розгортання стеку? У чому його відмінність від використання стеку при виклику звичайної функції?
3. Як за допомогою макросу `assert` можна швидко перевірити коректність програми?
4. Як у мові C++ передати дані з командного рядку?
5. Як забезпечити коректність даних, переданих з середовища операційної системи у програму на мові C++?

Комп’ютерний практикум №8
Використання зв’язних списків

Мета роботи: отримати навички реалізації абстрактних типів даних, динамічних структур даних та засвоїти основні принципи роздільної компіляції.

8.1. Теоретичні відомості

Побудова зв’язних списків

Зв’язний список — це структура даних або вузлів, кожний з яких містить як власні дані, так і один або два вказівника на наступний та/або попередній вузол. Принциповою перевагою таких конструкцій перед звичайним масивом є їх гнучкість, оскільки порядок елементів зв’язного списку може не збігатися з порядком фізичного розташування окремих елементів у оперативній пам’яті.

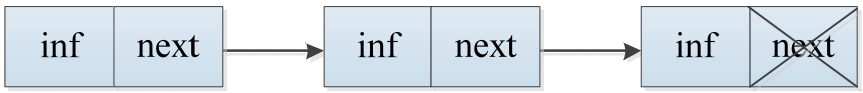
Зв’язний список — це різновид лінійних структур даних, що є послідовністю елементів, зазвичай відсортованих відповідно до деякого критерію. Послідовність може містити будь-яку кількість елементів, оскільки при створенні списку використовується динамічний розподіл пам’яті.

Кожний елемент зв’язного списку є окремим об’єктом (або структурою), що містить поля для зберігання інформації та вказівник на наступний

елемент списку. (Якщо список є двозв’язним, то в такому об’єкті зберігається також вказівник на попередній елемент.)

Схематично одно- та двозв’язний список можна представити наступним чином.

Однозв'язний список



Двозв'язний список

Рис. 8.1.

Пересування по списку здійснюється за вказівниками, які містять адресу наступного елемента списку. При додаванні до списку нового елемента необхідно динамічно виділити для нього пам’ять за допомогою оператора `new` та присвоїти відповідні адреси вказівникам сусідніх елементів.

Види зв’язних списків

Існує декілька можливих типів зв’язних списків. Найбільш поширені з них розглядаються у цьому розділі.

Однозв’язний список

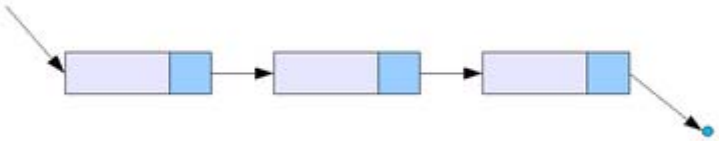


Рис. 8.2.

В однозв'язному списку можна пересуватися лише у напрямку від першого до останнього вузла. При цьому довідатися адресу попереднього елемента неможливо.

Двозв'язний список

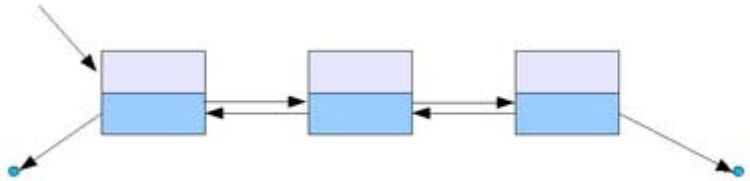


Рис. 8.3.

По двозв'язному списку можна пересуватися в будь-якому напрямку — як від першого елементу у напрямку останнього, так і у зворотному напрямку. У цьому списку простіше робити видалення та перестановку елементів, оскільки завжди відомі адреси попереднього та наступного елементів списку.

Кільцевий зв'язний список

Різновидом зв'язних списків є кільцевий (або циклічний, замкнений) список. Він теж може бути одно- або двозв'язним. Останній елемент кільцевого списку містить покажчик на перший елемент, а перший (для двозв'язного списку) — на останній.

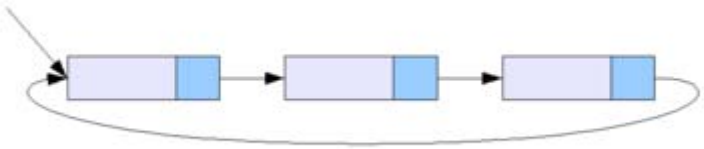


Рис. 8.4.

Роздільна компіляція

Оскільки реальні програми є досить складними сутностями, що підтримують розгалужену логіку функціонування, для реалізації модульного принципу їх розробки на мові C++ практично всі середовища розробки надають можливість створення та використання *проекту* програмного продукту або розміщення користувацьких функцій в окремих файлах. Тоді ці файли можна компілювати окремо, а потім на завершальному етапі за допомогою *компоновщика* (linker) або *редактора зв'язків* зв'язувати їх та необхідні бібліотеки разом в єдину виконувану програму. Якщо зміни були внесені тільки в один файл, то можна перекомпілювати тільки його та зв'язати з раніше відкомпільованими версіями інших файлів. Цей механізм полегшує роботу з великими програмами. У більшості середовищ розробки на мові C++ є додаткові засоби, що дозволяють у автоматичному режимі відслідковувати внутрішні залежності між файлами програми, а також те, коли ці файли було модифіковано останній раз. Якщо програма розроблюється в інтегрованому середовищі, такому як Microsoft C++, то доступ до таких можливостей можна отримати за допомогою команд меню `Project`. У середовищі Unix та Linux аналогічну функціональність надає утиліта `make`.

Водночас з роздільною компіляцією пов'язані також і деякі нові проблеми. Наприклад, якщо в двох різних функціях з двох різних файлів використовуються одні і ті самі оголошення структур, то такі оголошення необхідно розмістити в обох файлах. (В іншому випадку компілятор згенерує повідомлення про використання невідомого типу, оскільки ці файли компілюються окремо). Однак просте копіювання цих оголошень в усі файли, де вони застосовуються, може призвести до виникнення нових помилок. Навіть якщо оголошення структури буде скопійовано правильно, слід постійно пам'ятати, що зміни треба вносити в обидва файли, а це може легко призвести до помилок.

Ця проблема легко розв'язується за допомогою додаткового заголовного файлу. Достатньо додати до нього всі необхідні оголошення, а потім за допомогою директиви `#include` включити його у всі файли з програмним кодом, де ці оголошення використовуються. При такому підході для зміни оголошення структури достатньо буде зробити це один раз у відповідному заголовному файлі. Крім того, у заголовному файлі можна розміщати і інші програмні компоненти.

У найпростішому випадку проект програми може містити наступні файли:

- заголовний файл, що містить оголошення структур даних і прототипи функцій, що використовують ці структури;
- файл з програмним кодом функції `main()`, у якому викликаються ці функції;
- файл з визначеннями користувацьких функцій.

Така стратегія може бути успішно використана для організації програм. Наприклад, при розробці програми, що використовує раніше реалізовані функції, достатньо скористатися існуючим заголовним файлом і додати до складу відповідного проекту файл із реалізацією цих функцій.

У заголовних файлах не слід розміщати визначення функцій або оголошення змінних, оскільки при використанні такого файлу більше одного разу будуть порушені вимоги правила одного визначення. Зазвичай у заголовних файлах повинні використовуватись наступні елементи:

- прототипи функцій;
- символьні константи, визначені за допомогою директиви `#define` або специфікатора `const`;
- оголошення структур;
- оголошення класів;
- оголошення шаблонів;
- вбудовані функції.

В інтегрованих середовищах розробки програм не потрібно додавати заголовні файли до списку проекту (`project list`). Не слід використовувати директиву `#include` для включення одних файлів програмного коду в інші файли.

Більш докладно теоретичні питання, пов'язані з даною лабораторною роботою, викладено в [3].

8.2. Приклад

У наведеному нижче прикладі реалізовано простий зв'язний список, кожний елемент якого містить поле для зберігання даних та вказівник на наступний елемент. При цьому весь програмний код зв'язано на основі принципу роздільної компіляції. До складу проекту входить три файли: два перших складають пару «заголовний файл–файл з реалізацією», що забезпечує можливість легкого підключення бібліотеки функцій у будь-яку

програму на мові C++, третій містить реалізацію головної функції. Зверніть увагу, що в прикладі заголовний файл називається `module.h`

```
// FILE module.h
#include <iostream>
#include <string>

using namespace std;

struct element
{
  string str;
  element* next;
};

element* EnterList();
int Count(element*);

// FILE module.cpp with functions body
#include "module.h"

element* EnterList()
{
  element *first, *current;
  string answer;

  cout<< "enter first string : ";
  first=current=new element;
```

```
    cin>> current->str;
    cout<< "do you want new string? (n for exit)";

    cin>> answer;
    while(answer != "n")
    {
        current->next = new element;
        current = current->next;
        cout <<   "enter string :  ";
        cin>>current->str;
        cout<< "do you want to enter new string? (n for
exit)";
        cin>> answer;
    }

    current->next = NULL;
    return first;
}

  int Count(element* list)
  {
    int result = 0;

    if(!list)
    {
        cout << "List is empty";
    }

    while(list)
    {
```

```
        result++;
        list=list->next;
    }

    return result;
}

  // FILE project1.cpp with main() function
  #include "module.h"

  int main()
  {
    element *current, *top;
    top = EnterList();
    current = top;
    while(current!=0)
    {
        cout<< current->str << endl;
        current=current->next;
    }


    cout << endl << "Size of List is " <<
    Count(top) << endl;

    cin.get();
    cin.get();
    return 0;
  }
```

Для створення виконуваної програми потрібно додати до її проекту два файли з програмним кодом.

8.3. Порядок виконання роботи

1. Проаналізувати умову задачі.
2. Розробити алгоритм розв'язання задачі згідно з номером варіанту. Розробити блок-схему алгоритму.
3. Створити програму розв'язання задачі згідно з номером варіанту.
4. Результати роботи повинні бути представлені у вигляді протоколу та задовольняти нижченаведеним вимогам.

Пункти 2–3 *Порядку виконання роботи* повинні задовольняти наступним вимогам:

- у роботі повинні використовуватись типи даних користувача (структури), динамічні структури даних (перелік, черга тощо) та засоби роботи з вільною пам'яттю (оператори `new`, `delete` тощо);
- необхідно передбачити можливість вводу даних, їх модифікації, сортування та збереження; у разі використання вже створеного файлу (його завантаження) ті ж самі операції повинні виконуватись з ним;
- програму треба реалізувати з використанням модульного принципу програмування (шляхом створення проекту програми), коли всі функції, крім головної, містяться в декількох окремих файлах;
- передбачити можливість зберігання введеної інформації у файлі на диску; можливість завантаження даних з файлу потрібного формату на диску та запис у той же або новий файл модифікованих даних.

8.4. Варіанти завдань

Реалізувати варіанти завдань згідно з номером залікової книжки.

1. Одне з можливих представлень "довгого" тексту — це розділити його на ділянки (рядки) і створити список (додавши ознаку кінця тексту). Використовуючи дане представлення тексту, визначити функції:

а) *число рядків (T)* для підрахунку числа рядків у тексті *T*;

б) *елем (T, i, j, c)*, що перевіряє, чи є в тексті *T* рядок з номером *i*, і, якщо є, то присвоює *j*-тій літері цього рядка значення параметру *c*;

в) *перестановка (T, i, j)*, що міняє місцями *i*-й і *j*-й рядок тексту *T*;

г) *заміна (T, i, j)*, що замінює *i*-й рядок тексту *T* на копію *j*-го рядка.

д) *виведення (T)*, що порядково виводить текст *T*;

е) *введення (T)*, що зчитує з вхідного файлу послідовність літер до ознаки кінця тексту (`#`) і формує з них текст *T*. (Використовувати `fgets()`)

2. Одне з можливих представлень "довгого" тексту — це розділити його на ділянки (рядка) і створити список (додавши ознаку кінця тексту). Використовуючи дане представлення тексту, визначити функції:

а) *додати (T, i, j)*, що додає після *i*-го рядка тексту *T* копію *j*-го рядка;

б) *видалити (T, j)*, що видаляє *j*-й рядок з тексту *T*;

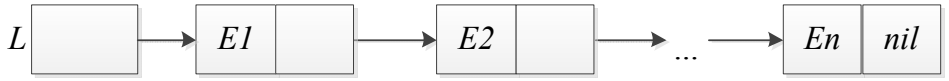
в) *пошук (T, c, i, j)*, що визначає, чи входить літера *c* у текст *T*, і, якщо входить, то присвоює параметрам *i* і *j* «координати» першого входження цієї літери, *i* — номер рядка, *j* — номер позиції в цьому рядку;

г) *максимум(T,c)* – номер рядка з максимальним числом раз входження літери *c*;

д) *виведення (T)*, що друкує порядкого текст *T*;

е) *введення (T)*, що зчитує з вхідного файлу послідовність літер до ознаки кінця тексту (`#`) і формуючу з них текст *T*. (Використовувати `fgets()`)

3. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а

б

```
typedef double TE ;
struct ланка {TE елем; ланка* наступна; };
```

При цьому параметри *L*, *L1* і *L2* позначають списки, а параметри *E*, *E1* і *E2*-дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність.

Визначити функції, що :

а) визначає, чи є список *L* порожнім;

б) знаходить середнє арифметичне елементів непорожнього списку *L* ;

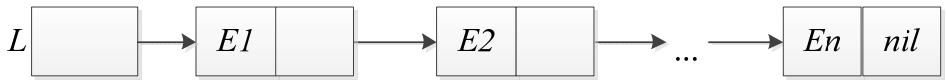
в) заміняє в списку *L* усі входження *E1* на *E2*;

г) міняє місцями перший і останній елементи непорожнього списку *L*;

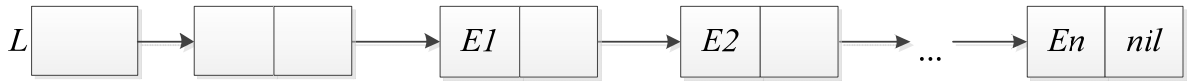
д) вводить дані списку з екрана ;

е) виводить список у файл.

4. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а



б

```
typedef char TE ;
struct ланка { TE елем; ланка* наступна; };
```

При цьому параметри *L*, *L1* і *L2* позначають списки, а параметри *E*, *E1* і *E2*-дані типи *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність.

Визначити функції, що:

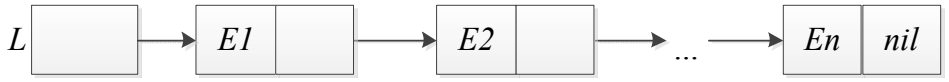
а) визначає, чи є список *L* порожнім ;

б) упорядковує елементи списку *L* за алфавітом ;

в) знаходить коди ASCII останнього і передостаннього елементів списку *L*, що містить не менш двох елементів.

г) виводить список у файл.

5. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а

б

```
typedef char TE [10];
struct ланка { TE елем; ланка* наступна; };
```

Параметри *L*, *L1* і *L2* позначають списки, а параметри *E*, *E1* і *E2*—дані типи TE, до яких можна застосовувати операції присвоювання і перевірки на рівність.

Визначити функції, що підраховують кількість слів списку *L*, що;

а) починаються і закінчуються однієї і тією же літерою;

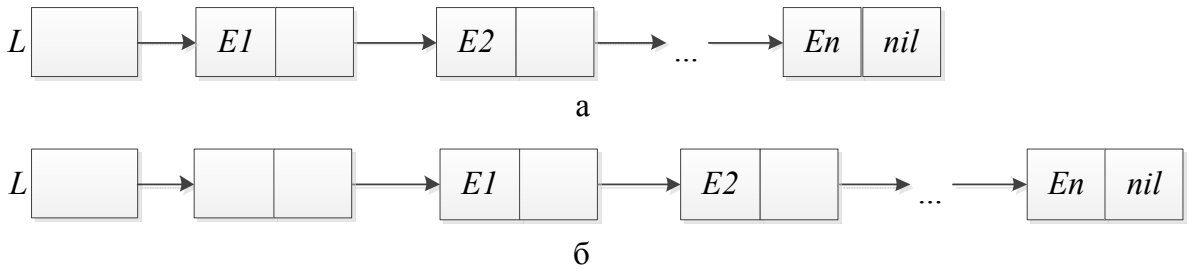
б) починаються з тієї ж літери, що і наступне слово;

в) збігаються з останнім словом;

г) зберігає список в окремий файл;

д) зчитує список з файлу.

6. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



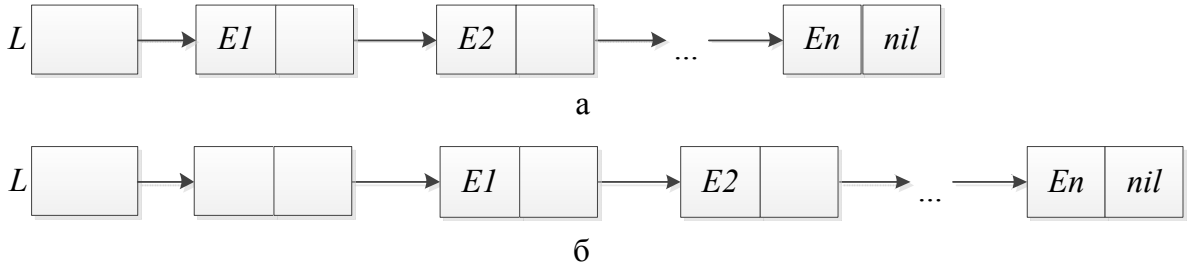
а

б

```
typedef char TE [10];
typedef file of TE;
масив array [1..50] of TE;
```

Визначити функцію, значенням якої є список, побудований з елементів :
а) файлу `f` ;
б) масиву `x` (список будувати від кінця).

7. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а

б

Параметри *L*, *L1* і *L2* позначають списки, а параметри *E*, *E1* і *E2*-дані типи *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність.

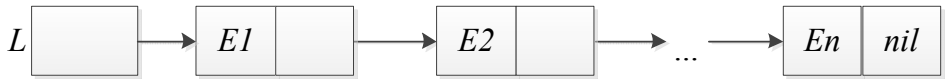
Визначити функцію, що
а) видаляє зі списку *L* перше входження елемента *E*, якщо таке є;

б) за списком *L* будує два нових списки: *L1* — з позитивних елементів і *L2* — з інших елементів списку *L {TE* = `double`*}*.

в) видаляє зі списку *L2* перший негативний елемент, якщо такий є,

г) виводить результат у два різних файли (використовується одна функція).

8. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а

б

Параметр *L* позначає список, а параметр *E*, *E1* — дані типу *TE*, до яких можуть застосовуватись операції присвоювання і перевірки на рівність. *{TE* = `char*`*}*

Описати процедуру, що

а) вставляє в початок списку *L* новий елемент *E*;

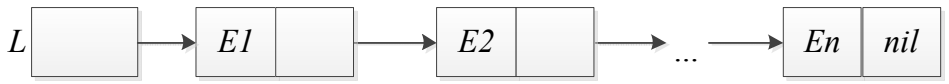
б) у кінець списку *L* новий елемент *E*;

в) новий елемент *E* після першого елемента непорожнього списку *L*;

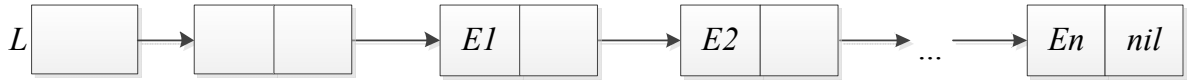
г) у список *L* новий елемент *E1* за кожним входженням елемента *E* ;

д) записує список у текстовий файл.

9. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а



б

Параметр *L* позначає список, а параметр *E*, *E1* — дані типу *TE*, до яких можуть застосовуватись операції присвоювання і перевірки на рівність. *{TE = char*}*

Описати процедуру, що

а) додає в список *L* новий елемент *E1* перед першим входженням елемента *E*, якщо *E* входить у *L*;

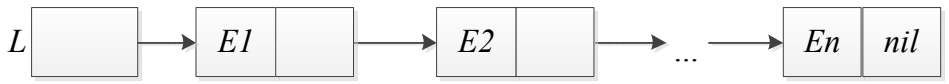
б) у непорожній список *L* додає пари нових елементів *E1* і *E2* перед його останнім елементом;

в) у непорожній список *L*, елементи якого упорядковані в порядку неспадання, новий елемент *E* так, щоб збереглася упорядкованість ;

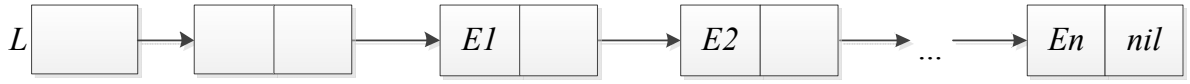
г) переносить у початок непорожнього списку *L* його останній елемент ;

д) записує список у текстовий файл.

10. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а



б

Параметр *L* позначає список, а параметри *E*, *E1* — дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність. *{TE =* `char*`*}*

Визначити функції, що видаляють :

а) з непорожнього списку *L* перший елемент;

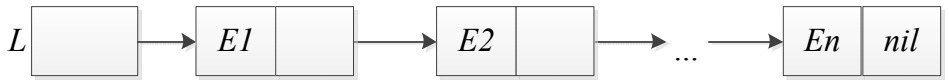
б) зі списку *L* другий елемент, якщо такий є;

в) зі списку *L* за кожним входженням елемента *E* один елемент, якщо такий є і він відмінний від *E*

г) з непорожнього списку *L* останній елемент;

д) виводить список у текстовий файл.

11. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а

б

Параметр *L* позначає список, а параметри *E*, *E1* — дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність. *{TE =* `int`*}*

Непорожня послідовність натуральних чисел вводиться з екрану, за якої випливає 0. Визначити функції, що :

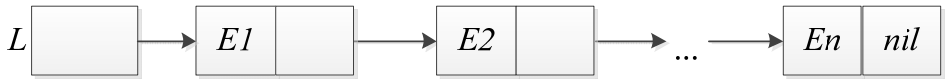
а) виводить числа в зворотному порядку ;

б) виводить порядкові номери тих чисел послідовності, що мають найбільшу величину;

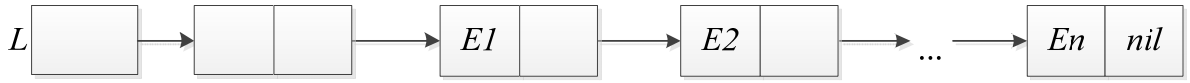
в) сортує числа в порядку їхнього неспадання;

г) записує список у файл.

12. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а



б

Параметр *L* позначає список, а параметри *E*, *E1* — дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність. *{TE =* `char*`*}*

Визначити функцію, що:

а) перевіряє на рівність списки *L1* і *L2*;

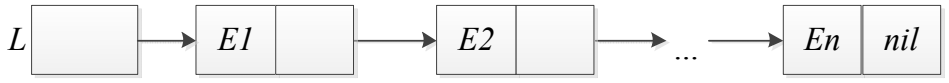
б) визначає, чи входить список *L1* у список *L2*;

в) перевіряє, є чи в списку *L1* (*L2*) хоча б два однакових елементи ;

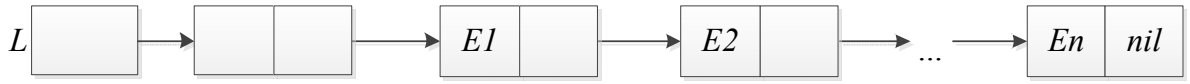
г) переносить у кінець непорожнього списку *L1* (*L2*) його перший елемент ;

д) виводить результат в окремий файл.

13. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а



б

Параметр *L* позначає список, а параметри *E*, *E1* — дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність. *{TE =* `char`*}*

Визначити функції, що:

а) додає в кінець списку *L1* всі елементи списку *L2*;

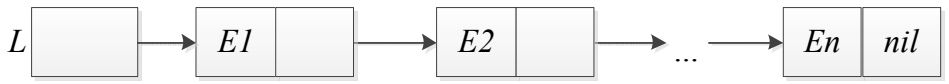
б) вставляє в список *L1* за першим входженням елемента *E* всі елементи списку *L2*, якщо *E* входить у *L1*;

в) перевертає список *L1* (*L2*), тобто змінює посилання в цьому списку так, щоб його елементи виявилися розташованими в зворотному порядку;

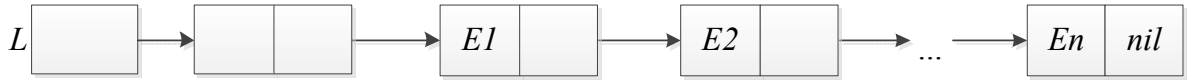
г) у списку *L* з кожної групи підряд йдуть рівних елементів залишає тільки один;

д) виводить список у файл.

14. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а



б

Параметр *L* позначає список, а параметри *E*, *E1* — дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність. *{TE =* `double`*}*

Визначити рекурсивні функції, що:

а) визначає, чи входить елемент *E* в список *L*;

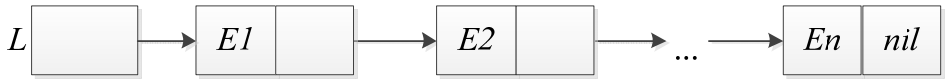
б) підраховує число входжень елемента *E* в список *L*;

в) знаходить максимальний елемент непорожнього списку *L*;

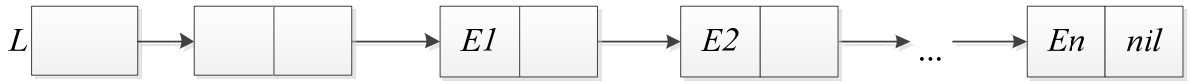
г) заміняє в списку *L* всі входження *E1* на *E2*;

д) виводить список у файл.

15. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а



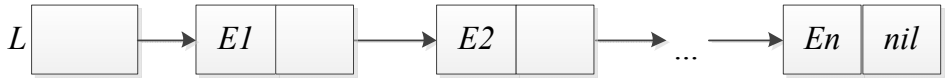
б

Параметр *L* позначає список, а параметри *E*, *E1* — дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність. *{TE =* `double`*}*.

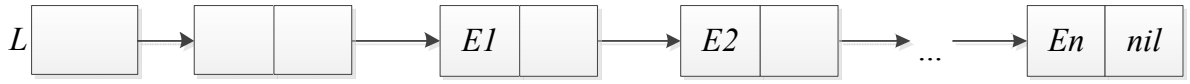
Визначити рекурсивні функції, що :

а) видаляє зі списку *L* усі входження елемента *E* ;
б) будує *L1*—копію списку *L* ;
в) подвоює кожне входження елемента *E* в список *L*;
г) залишає в списку *L* тільки перші входження однакових елементів.
д) виводить список у файл.

16. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а



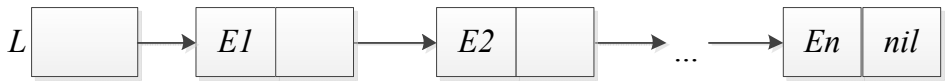
б

Параметр *L* позначає список, а параметри *E*, *E1* — дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність. *{TE =* `double`*}*

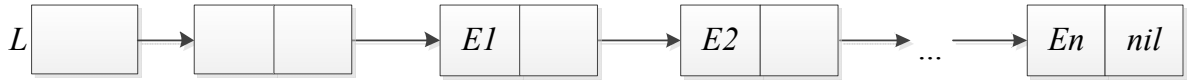
Визначити функції, що формують список *L*, включивши в нього по одному разу елементи, що:

а) входять хоча б в один зі списків *L1* і *L2*;
б) входять одночасно в обидва списки *L1* і *L2*;
в) входять у список *L1*, але не входять у список *L2*;
г) входять в один зі списків *L1* і *L2*, але в той же час не входять в інший з них.
д) виводить список *L* в окремий файл.

17. Використовувати (лінійні) односпрямовані списки без заголовної ланки (мал. а) або з заголовною ланкою (мал. б) при наступному їхньому описі



а



б

Параметр *L* позначає список, а параметри *E*, *E1* — дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність. *{TE = double}*

Визначити функції, що поєднують два упорядкованих за зростанням списку *L1* і *L2 (TE = double)* в одному упорядкований за зростанням список;

а) побудувавши новий список *L*;

б) змінюючи відповідним чином посилання в *L1* і *L2* і привласнивши отриманий список параметрові *L1*.

в) результат записується в окремий файл.

18. Параметр *L* позначає список, а параметри *E*, *E1* — дані типу *TE*, до яких можна застосовувати операції присвоювання і перевірки на рівність. Список має структуру

```
typedef ланка* слово;
struct ланцюжок { char* буква; ланка* зв'язок; };
```

Визначити функцію, що:

а) у списку *L* переставляє місцями перші й останнє непорожні слова, якщо в *L* є хоча б два непорожніх слова;

б) друкує текст із перших букв усіх непорожніх слів списку *L*

в) видаляє з непорожніх слів списку *L* їхні перші букви;

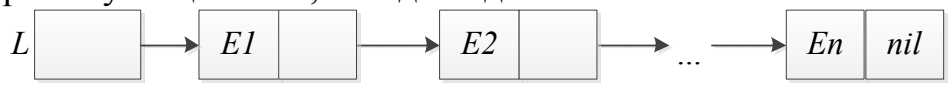
г) друкує всі непорожні слова списку *L*;

д) визначає кількість слів у непорожньому списку *L*, відмінних від останнього.

е) виводить результат в окремий файл.

19. Багаточлен $P(X) = A_n * X_n + A_{n-1} * X_{n-1} + ... + A_1 * X + A_0$

з цілими коефіцієнтами можна представити у виді списку (мал. 14 а), причому якщо Ai =0, та відповідна ланка



а

б

не включається в список (на мал. 14 б показане представлення багаточлена $P(X) = 52 * X^{40} - 3 * X^8 + X$.

Описати тип даних, що відповідає такому представленню багаточленів, і визначити наступні функції для роботи з цими списками -багаточленами

а) *логічну функцію дорівнює(p,q)*, що перевіряє на рівність багаточлени *p* і *q*;

б) *функцію знач (p, x)*, що обчислює значення багаточлена *p* у цілочисельній точці *x*

в) *процедуру диф (p, q)*, що будує багаточлен *p* — похідну багаточлена *q*;

г) *процедуру висновок (p, v)*, що друкує багаточлен *p* як багаточлен від змінної, однобуквене ім'я якої є значенням літерного параметра *v* і; (наприклад, для зазначеного вище багаточлена S процедура *вивід(s',y')* повинна надрукувати *52y**40-3y**8+y)*;

д) *процедуру введення (p)*, яка зчитує з вхідного файлу безпомилковий запис багаточлена (за нею — пробіл) і формує відповідний список-багаточлен *p*.

20. Нехай *L* позначає кільцевий (циклічний) двонаправлений список із заголовною ланкою (мал. 15) (з елементами *E* типу `*char`)

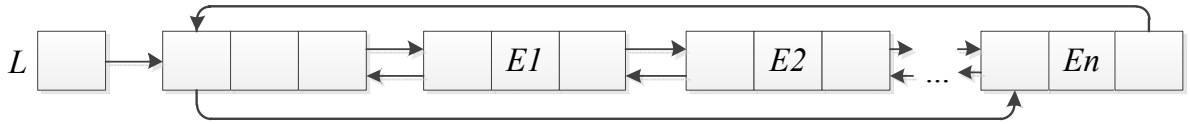


Рис.15

Визначити функції, що:

а) визначає, чи є список *L* порожнім;

б) підраховує кількість елементів списку *L*, у яких рівні «сусіди»;

в) визначає, є чи в списку *L* хоча б один елемент, що дорівнює наступний за ним (по колу) елементові;

г) у списку *L* переставляє в зворотному порядку всі елементи між першим і останнім входженнями елемента *E*, якщо *E* входить у *L* не менш двох разів;

д) виводить список у файл.

8.5. Контрольні запитання

1. Які переваги та недоліки мають динамічні структури даних?
2. Що таке зв'язний список? Їх типи.
3. Базові принципи побудови зв'язних списків.
4. У чому полягає модульний принцип побудови програм на мові C++ на рівні організації програмного коду?
5. Переваги та недоліки роздільної компіляції.

Список літератури

1. Пол А. Объектно-ориентированное программирование на C++. — Бином, 2001. — 464 с.

2. Страуструп Б. Язык программирования C++. Специальное издание. — Бином, 2008. — 1104 с.

3. Стивен Прата - Язык программирования C++. Лекции и упражнения. — Вильямс, 2007. — 1184 с.

4. Х. М. Дейтел, П. Дж. Дейтел - Как программировать на C++. — Бином, 2007. — 800 с.