МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Програмування
Структурний підхід
Лабораторний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітніми програмами «Математичні методи криптографічного захисту інформації» та «Математичні методи моделювання, розпізнавання образів та безпека даних»
спеціальності 113 «Прикладна математика»

Київ
КПІ ім. Ігоря Сікорського
2021

Програмування. Структурний підхід. Лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 113 «Прикладна математика» / Н. М. Куссуль, А. Ю. Шелестов, А. М. Лавренюк, Л. В. Булигіна; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1501 Кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 113 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 6 від 25 лютого 2021 р.) за поданням Вченої ради Фізико-технічного інституту Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (протокол № 1 від 11 січня 2021 р.)

Електронне мережне навчальне видання

Програмування
Структурний підхід
Лабораторний практикум

Укладачі: *Куссуль Наталія Миколаївна,* д. техн. наук, проф.
Шелестов Андрій Юрійович, д. техн. наук, проф.
Лавренюк Алла Миколаївна, канд. техн. наук, доц.
Булигіна Людмила Вікторівна, асис.

Відповідальний редактор *Смирнов С.А.,* к.ф.-м.н., доц.

Рецензент *Тітков Д. В.*, старший викладач кафедри інформаційної безпеки ФТІ, Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського"

Навчальний посібник «Програмування. Структурний підхід. Лабораторний практикум» присвячено вивченню мови програмування C++ студентами за спеціальністю 113 «Прикладна математика». Метою є отримання навичок розробки програмного забезпечення на основі структурного підходу з застосуванням модульного принципу програмування, різних типів даних та змінних, використання керуючих операторів та функцій, засобів прямого доступу до пам’яті (вказівників) та роботи з файлами. При вивченні цієї навчальної дисципліни студенти засвоять також інструменти роботи з файлами, обробки строкової інформації та використання потужних засобів бібліотеки STL (Standard Template Library). Посібник містить необхідний теоретичний матеріал, приклади програм, а також завдання для виконання лабораторного практикуму.

ЗМІСТ

Лабораторна робота №1
Принципи створеня програм на мові С++

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

1.1. Теоретичні відомості

Структура програми

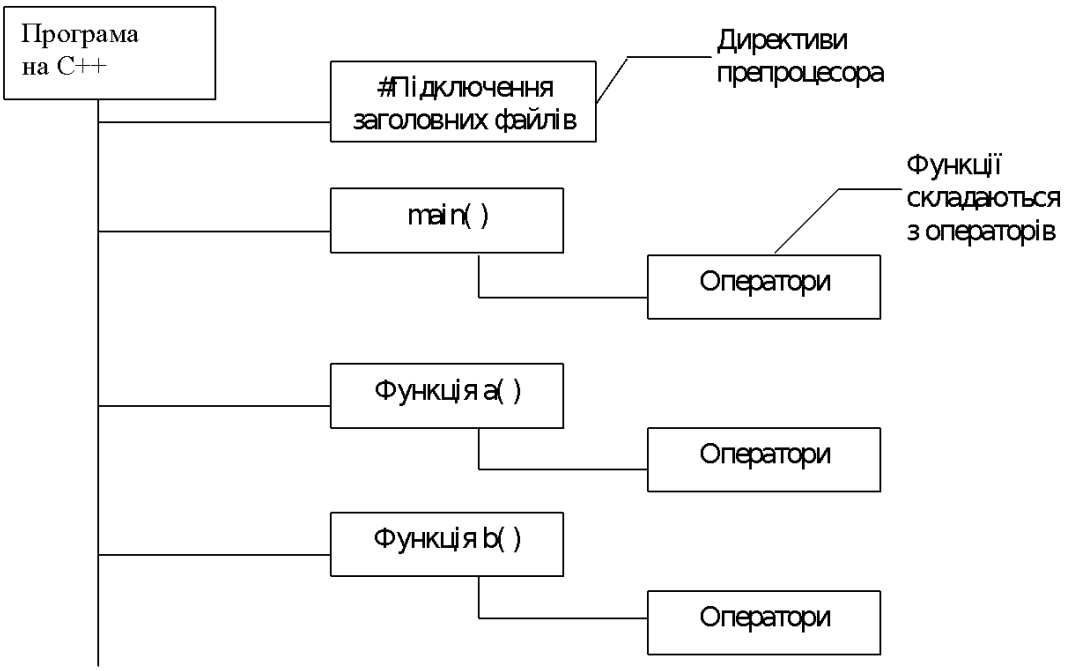
Програма на мові С++ є блочно-структурованою та, як правило, містить деякий набір функцій. *Функція* — це іменована частина програми, до якої можна звертатися з її інших частин шляхом вказання імені, наприклад, SimpleFunc(5, 1.9). У даному випадку функція має ім'я SimpleFunc, якій передається два числових параметра.

Питання створення та використання функцій більш докладно розглядаються у лабораторній роботі № 3.

Кожна програма на мові С++ у своєму складі повинна мати *головну функцію* **`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. Для того, щоб функція була доступна, необхідно, щоб до її виклику про неї було відомо компілятору.

Основна структура програми на мові С++ має наступний вигляд.



Нижче наведено приклади двох простих програм на мові С++.

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";            //виведення рядка на екран
  cin.get();               // очікується натискання клавіші <Enter>
  return 0;                // оператор повернення завершує функцію
                           // main()
}                           //кінець тіла функції main()
```

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;
  cin>>a;                //потік введення
  cout<<“Ви ввели число ”<<a<<endl; //потік виведення
  return 0;  //оператор повернення завершує функцію main()
}
```

endl — це маніпулятор виведення, що вставляє у вихідний потік символ переходу (“/n”) на новий рядок.

Засоби введення/виведення

При запуску програми на мові C++ автоматично створюється декілька стандартних потоків, а саме, наступних: cin (стандартний потік вводу з клавіатуры), cout (стандартний потік виводу на екран). Існує ще 2 стандартних потоки cerr та clog, що призначені для виводу помилок. Для того щоб зкористуватися потоками, достатньо підключити заголовний файл <iostream> та вказати стандартний простір імен.

```
#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 C++. Більш докладно можливості їх використання будуть розглядатися у лабораторній роботі № 5 та у курсі **Программування. Об'єктно-орієнтований підхід**. Крім потоків для вводу/виводу інформації можна використовувати також функціїї мови C, зокрема функцію printf() та scanf(). Більш докладно ці засоби також будуть розглядатися у лабораторній роботі № 5.

Етапи створення виконуваного коду програми

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

1. Зкориставшись текстовим редактором, написати програму і зберегти її у файлі на диску. Цей файл містить вихідний код програми.
2. Скомпілювати вихідний код.

Цей процес є двошаговим. Зокрема, при компяліції програми на мові C++ спочатку виконується препроцесинг, 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, зкомпонувати єдину програму. Програми C++ 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'єктними кодами 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 (препроцесинг) вихідного коду. При цьому всі директиви препроцесора, що починаються з символа #, оброблюються 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 препроцесингу, 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 C++, 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 файла міститься у таких дужках, то пошук файлу буде проводитися у спеціальному каталозі заголовних файлів (як правило, це каталог INCLUDE).
Для підключення власностворених 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 C++, такі як, 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 вихідного 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 буд-якому випадку, для отримання довідкової інформації щодо параметрів командного рядка компілятора можно зкористатись параметром /? або –h.
Якщо компіляція пройшла успішно, буде створено виконуваний файл unit1.exe.

Операції і вирази в C++

Вираз — це послідовність знаків операцій, операндів і круглих дужок, що задає обчислювальний процес знаходження результату певного типу. *Найпростішим виразом* є константа, змінна або виклик функції.
Операції і вирази мови C++ дозволяють задати певну послідовність дій. *Прості вирази* містять символ операції та операнди. Прикладом простого виразу з *бінарною* операцією є віднімання: **`3.4 – x`**.

Операнд являє собою елемент-учасник операції. Операндами можуть бути *константи, змінні, виклики функцій* і *вирази.*

У мові С++ визначено наступні основні арифметичні операції.

+, -, *, / — знаки *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 дійсне число). Тип операції діленя, 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 (можливо порожній). Прикладом виразу з викликом функції може біти наступним.

```
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. Пріоритети операцій подані в табл. 1.

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.

У мові С++ підтримуються також дві унарні операції інкременту та декременту **++** і **--** для збільшення і зменшення на одиницю значення операнда відповідно. Їх можна вказувати як перед операндом, так і після нього. У першому випадку (**++x** або **--x**) значення операнда **x** змінюється перед його використанням у виразі, а у другому (**x++** або **x--**) — після його використання. Наприклад:

```
b=b1=3; c=c1=5;a=b+c++; /* b=3; c=5+1=6; a=3+5=8 */
a1=b1+(++c1);/* b1=3; c1=5+1=6 ; a1=3+6=9 */
```

<, **>**, **==**, **>=**, **<=**, **!=** — це символи *операцій відношення* (менше, більше, дорівнює, більше або дорівнює, менше або дорівнює, не дорівнює), які використовуються в *логічних виразах;*

&&, **||**, **!** — *знаки логічних операцій* «І», «АБО» і «НЕ»;

&, **|**, **^**, **<<**, **>>**, **~** — знаки *порозрядних логічних операцій,* які виконуються над однойменними бітами машинного слова: «І», «АБО», виключаюче «АБО», зсув вліво, зсув вправо, інверсія;
?, **:** — знаки, які використовуються в *умовній тернарній операції.* Наприклад, результатом виразу **(a>0) ? a : 0** буде значення **a**, якщо **a>0** і нульове значення — в іншому випадку.
***, &** — знаки *адресних операцій.*
[і **]** — квадратні дужки використовуються при роботі з масивами для задання розмірів масиву і доступу до елементу масиву;
(і **)** — дужки використовуються для зміни пріоритету виконання операцій у виразах і при вказівці аргументів функцій;
Зручно використовувати також наступні *операції присвоювання:*
***=**, **/=**, **%=**, **+=**, **-=**, **<<=**, >>=, **&=**, **^=**, **|=**
Вони застосовуються для компактного запису *операторів присвоювання.* Наприклад, при виконанні двох операторів **a=a+b** та **a+=b** буде отримано однаковий результат.
Оператори закінчуються крапкою з комою. Прості оператори можна об'єднати в складений або *блок* за допомогою фігурних дужок, за якими крапку з комою можна не ставити.

Таблиця 1. **Пріоритети, порядок виконання та призначення операцій у мові С++**

Пріо-ритет	Знаки операцій	Назви операцій	Порядок виконання
1.	. ->[] () ++ --	вибір елемента за іменем вибір елемента за вказівником вибір елемента за індексом виклик функції або конструювання значення постфіксний інкремент постфіксний декремент	зліва-направо
2.	sizeof ++ -- ~ ! + - & new delete (ім'я_тип	розмір операнда в байтах префіксний інкремент префіксний декремент інверсія (порозрядне заперечення) логічне заперечення унарний плюс унарний мінус адреса виділення пам'яті або створення звільнення пам'яті або знищення перетворення типу	справа-наліво

	у)		
3.	.* ->*	вибір елемента по імені через вказівник вибір елементу по вказівнику через вказівник	зліва-направо
4.	* / %	множення ділення остача від ділення цілих (ділення по модулю)	зліва-направо
5.	+ -	додавання віднімання	зліва-направо
6.	<< >>	зсув вліво зсув вправо	зліва-направо
7.	< > <= >=	менше більше менше або дорівнює більше або дорівнює	зліва-направо
8.	== !=	дорівнює не рівне	зліва-направо
9.	&	порозрядне І	зліва-направо
10.	^	порозрядне виключаюче АБО	зліва-направо
11.	|	порозрядне АБО	зліва-направо
12.	&&	логічне І	зліва-направо
13.	||	логічне АБО	зліва-направо
14.	?:	умовна операція	справа-наліво
15.	= *=, /=, %=, +=, -=, &=, ^=, |=	присвоювання (просте і складене)	справа-наліво
16.	*throw*	генерація виключення	справа-наліво
17.	,	послідовність виразів	зліва-направо

Типи даних

Під *типом даних* (data type) розуміють множину припустимих 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`
- `wchar_t(wide character)`
- `void`

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

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

Перелічимо прості типи даних (від найкоротшого до найдовшого за кількістю інформації). Основні характеристики типів даних мови C++ наведено у таблиці 2.

`bool, char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float, double, long double`.

Таблиця 2. **Характеристики основних типів даних мови C++**

Ім'я типу	Розмір пам'яті, байтів (16/32-розрядна)	Діапазон значень для 16-розрядної архітектури	
[signed] char	1	-128	127
unsigned char	1	0	255
wchar_t	2	0	65 535
[signed] short [int]	2	-32 768	32 767
unsigned short [int]	2	0	65 535
[signed] int	Машинне слово	-32 768	32 767
unsigned int	Машинне слово	0	65 535
[signed] long [int]	4	-2 147 483 648	2 147 483 647
[unsigned] long [int]	4	0	4 294 967 295
float	4	3.4e-38	3.4e38
double	8	1.7e-308	1.7e308
long double	10	3.4e-4932	3.4e4932

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`).

Логічний тип bool

Логічні змінні типу **bool** можуть приймати одне з двох значень: *false* (хиба) та *true* (істина). За замовчуванням цілочисельне значення *false* рівне 0, а *true* рівне 1 (в загальному випадку true все крім 0). Логічні змінні широко використовуються в операціях порівняння, логічних операціях і логічних виразах. Розмір змінної залежить від реалізації, але звичайно складає 2 байти. Приклад оголошення:

```
bool reload = false,  in_range = true;
```

Приклад 1.3. Логічний тип `bool`

```
bool b1=a==b // якщо a=b, b1=true, інакше b1=false
bool b=7; // b= true
int i=true; // i=1
```

Тип `bool` можна перетворити в тип `int` і навпаки.

В арифметичних і логічних виразах логічні змінні перетворюються в цілі, і над ними виконуються операції.

Символьний тип char майже завжди займає 1 байт (знаковий або беззнаковий - залежить від реалізації). Кожна символьна константа відповідає певному числовому значенню (від 0 до 255) у відповідності до таблиці ASCII.

Приклад 1.4.1 Тип `char`

```
char c;
cin >>c;
```

Символьні типи є інтегральними. До них можна застосувати арифметичні й логічні операції, в них можна зберігати числові значення.

При використанні логічних та символьних значень у арифметичних виразах вони перетворюються у чисельні значення.

Приклад 1.4.2

```
char a = 'a';
char b = 98; //b = 'b', char b = 256 помилка
cout << a + 2; // вивід на екран 'c'
```

Цілий тип int завжди є знаковим.

```
int=signed int
```

Тип int залежить від конкретної платформи, тому цілочисельні літерали краще записувати в десятковій формі (8-ва та 16-ва форми запису можуть призвести до виникнення проблем).

Приклад 1.5 Тип int

```
0xffff = 65535
```
// в 32-розрядному комп'ютері

До типів із плаваючою точкою відносяться наступні.

- `float` - одинарної точності
- `double` - подвійної точності
- `long double` - розширеної точності

Приклад 1.6 Тип із плаваючою точкою

Літерали із плаваючою точкою:

```
1.23   0.2   3.23   1.0
1.23e-15     1.2e10
```

Тип void

Тип `void` (порожній) синтаксично поводиться як основний тип. Однак використати його можна тільки як частину похідного типу, об'єктів типу `void` не існує. Він використовується для того, щоб вказати, що функція не повертає значення, або як базовий тип для вказівників на об'єкти невідомого типу.

Приклад 1.7 Тип void

```
void f(); //функція не повертає значення
void * p; //вказівник на об'єкт невідомого типу
void a; //помилка
```

Перетворення типів

Перетворення типів від одного до іншого відбувається при присвоюванні і в змішаних виразах.

```
float d=3; //дані типу int перетворюються в тип float
```

В змішаних виразах відбувається два види автоматичного перетворення типів:

1) типи `bool`, `char`, `unsigned char`, `signed char`, `short` перетворюються в тип `int`;

2) якщо після першого кроку вираз має змішаний тип, то відповідно до ієрархії типів операнд з більш вузьким диапазоном перетворюється в операнд з більш широким диапазоном. При цьому загальне значення виразу відповідає більш широкому типу.

Операція зведення типів полягає у явному перетворенні даних одного типу в дані іншого типу. Ця операція є одномісною та має той же пріоритет, що й унарна операція. Вона наступний вигляд.

```
(Тип)x , де x змінна або Тип(x)
```

Приклад 1.8 Операція зведення типів

```
int i;
```

`(float)i+1` рівносильно `float(i)+ 1`

Зведення типів є коректним, якщо воно приводить до розширення типу, і некоректним, якщо воно приводить до звуження типу.

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 в***изначенні*** 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`).

Приклад 1.9 Оголошення змінних

```
…
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”;
char *season[]={“spring”, “summer”, “autumn”, “winter”}
typedef complex point;
double  sqrt(double);
…
```

Області видимості змінних

При оголошенні змінних у програмі велике значення має те місце, де вони оголошені. Від того, де оголошена змінна, залежать можливість її використання.
У C++ можливі три місця оголошення змінних.
По-перше, поза будь-яких функцій, у тому числі і головної функції `main()`. Така змінна називається **глобальною** і є **видимою** від місця оголошення **до кінця файлу**.
По-друге, змінна може бути оголошена всередині блоку, у тому числі й всередині тіла функції. Оголошена в такий спосіб змінна називається **локальною** і є **видимою до кінця блоку**. Така змінна поза блоком, у якому вона оголошена, невідома.

По-третє, змінна може бути оголошена як параметр функції. Крім спеціального призначення, а саме для передачі даних у функцію, параметр по відношенню до функції слід розглядати як її локальну змінну.
Таким чином, **в С++ існують 2 області видимості (2 контексти видимості) змінних: блок і файл.**

Приклад 1.10. Оголошення змінних. Програму для обчислення суми k чисел

```
#include <iostream.h>
void sum(int );  // прототип функції
int s=0;    // глобальна змінна
void main()
{
int i,b,k;    // локальні змінні
cout<<"\nВведіть кількість доданків";
cin>>k;
for(i=0;i<k;i++)
{
cout<<"\nВведіть новий доданок ";
cin>>b;
sum(b);// виклик функції
}
cout<<"\ns="<<s;
}

void sum(int c)
{
s=s+c;
}
```

У цій програмі змінна `s` є глобальною, вона доступна з обох функцій програми — `main()` і `sum()`, а змінні `i`, `b`, `k` та `c` — локальні, доступні тільки в тих функціях, де вони оголошені.
Якщо глобальна й локальна змінні мають одне і теж ім'я, тоді вважається, що оголошені дві різні змінні зі своїми областями використання. При цьому локальна змінна буде видима в тій функції, де вона оголошена, а глобальна у всій програмі за виключенням функції, у якій оголошена локальна змінна.

Час життя об'єкта даних

Об'єкт створюється, коли зустрічається його визначення і знищується, коли його ім'я виходить із області видимості.
Оголошення використовуються для визначення інтерпретації, що надається кожному ідентифікатору (імені). Це означає, що потрібно задати тип ідентифікатора щоб повідомити компілятор, до якого виду об'єктів відноситься ім'я.
Приклад 1.11 Визначення змінної

```
char symbol;
int number = 1;
double  sqrt;
```

Глобальні змінні створюються та ініціалізуються (тільки) один раз й "живуть" до завершення програми (**час життя глобальної змінної – до кінця роботи програми**). Об'єкти, визначені описом із ключовим словом `static`, поводяться так само.
Не ініціалізована явно статична (`static`) змінна неявно ініціалізується нулем.
Час життя локальної змінної – до виходу з блоку.

Приклад 1.12 Ініціалізація змінної

```
     int a = 1;
  void f()
  {
       int b = 1;      // ініціалізація b відбувається з
                       //кожним викликом функції f()
       static int c = 1;   // статична змінна створюється тільки
                           //один раз
       cout << " a=" << a++
            << " b=" << b++
            << " c=" << c++ << " \n"; }
int main() { while (a < 4) f(); return 0;}
```

Результат виконання програми виглядає так:

```
 a = 1 b = 1 c = 1
 a = 2 b = 1 c = 2
 a = 3 b = 1 c = 3
```

Приклад 1.13 Помилки

```
int a;
int a; // помилка! Повторне оголошення
extern   int  error_number;
extern  short  error_number; // невідповідність типів
```

Часом життя і областю видимості змінних можна керувати двома шляхами:
1) місцем оголошення змінної у програмі;
2) використанням модифікаторів **`auto, register, static, extern.`**
Автоматична (**`auto`**) змінна або константа має *локальну* область дії і відома тільки всередині блоку, у якому вона визначена. Для автоматичної змінної виділяється тимчасова пам'ять (стек). Пам'ять виділяється при вході в блок. При виході з цього блоку пам'ять, виділена для змінної, стає знову вільною.

Якщо специфікатор класу пам'яті не визначений, то зміна вважається автоматичною.

Регістрова (**register**) змінна відрізняється від автоматичної лише пам'яттю, що виділяється для її збереження. Регістрова змінна зберігається в регістрі процесора, і, відповідно, доступ до цієї змінної набагато швидший, ніж до змінної, яка зберігається в оперативній пам'яті (**auto**). У випадку відсутності вільних регістрів регістрова змінна стає автоматичною.

Зовнішня (**extern**) змінна є глобальною змінною. Специфікатор **extern** інформує компілятор, що змінна буде оголошена (без **extern**) в іншому файлі, де їй і буде виділена пам'ять.

Статичній (**static**) змінній (константі) пам'ять виділяється після її оголошення і зберігається до кінця виконання програми. Статичні змінні при оголошенні ініціалізуються нульовими (логічні, цілі і дійсні) порожніми значеннями.

Правило одного визначення

У мові C++ для кожного об’єкта (змінної, функції та ін.) повинно бути рівно одне визначення. Оголошень може бути декілька, але всі вони повинні бути погоджені за типом.

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;
```

Якщо значення умовного виразу істинне (не нуль), то виконується **оператор1**; якщо — хибне (рівне нулю), то **оператор1** пропускається, а виконується **оператор2**, що стоїть 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 **{}**.

Приклад. Знайти мінімум з двох чисел x та y.

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

…

Приклад. Перевірка коректності вводу змінної, яка повинна знаходитися у діапазоні від 1 до 31.

…

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

…

Приклад. Пошук максимуму з трьох чисел *a, b, c.*

…

```
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 частка (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:
1. 1111 із двійкової: $1111_2 = 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 = 15_{10}$
2.16 з вісімкової: $16_8 = 1 * 8^1 + 6 = 14_{10}$
3. FF із шістнадцяткової: $FF_{16} = 15 * 16^1 + 15 = 255_{10}$

1.2. Порядок виконання роботи

1.3.1. Проаналізувати умову задачі.
1.3.2. Розробити алгоритм та створити програму розв'язання задачі згідно з номером варіанту.
1.3.3. Результати роботи оформити протоколом.

1.3. Варіанти завдань

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.4. Завдання до додаткового теоретичного матеріалу

1) Знайти двійкове, вісімкове та шістнадцяткове представлення десяткового числа 31.

$31_{10} = 11111_2$, $31_{10} = 37_8$, $31_{10} = 1F_{16}$.

2) Знайти шістнадцяткове представлення десяткового числа 1000.

$1000_{10} = 3E8_{16}$.

3) Знайти десяткове представлення двійкого числа 11001.

$11001_2 = 25_{10}$

4) Знайти десяткове представлення двійкого числа 01000.

$01000_2 = 8_{10}$

1.5. Контрольні запитання

1. Яка структуру має програма на мові C++?
2. В чому полягають кроки перетворення програмного кода на мові C++ 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 C++?
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 C++ призначені для вводу/виводу даних?

Лабораторна робота №2
Умовна операція, множинний вибір та оператори циклів

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

2.1. Теоретичні відомості

Операція умови ?:

В мові C++ є короткий засіб запису оператора **if... else**. Для цього використовують тернарну операцію умови. Вона має наступну форму запису:

(умовний вираз) ? вираз1 : вираз2

Якщо умовний вираз істинний, то виконується **вираз1**, якщо хибний — **вираз2**.

Приклад 1. Знайти максимум з двох чисел x і y.

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

Операцію умови зручно використовувати у випадках вибору значення з двох можливих. Вирази з використанням операції умови більш компактні — це і є єдина перевага.

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; break;
}
```

Оператор вибору працює наступним чином. Спочатку підраховується вираз, який стоїть у дужках після **switch**. Далі виконується перехід на одну з міток, позначену 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** (її використання не є обов'язковим).

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**.

Типи операторів циклів

При виконанні програми часто виникає необхідність неодноразового повторення однотипних обчислень над різними даними. Для цих цілей використовують так звані цикли.

Цикл представляє собою частину програми, у якій одні й ті самі обчислення реалізуються 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 типу *while*

Цикл типу **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 типу **while** має наступну форму запису:

```
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 типу **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.

Приклад 2. Цикл типу **while**.

Користувачу дається 10 спроб щоб вгадати задане програмою 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 типу `do – while`

Цикл типу **`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 типу **`do – while`** дуже схожий на цикл типу **`while`**; різниця тільки в тому, що перевірка істинності виразу в циклі **`do – while`** має місце після виконання тіла циклу. Цей цикл має наступну форму запису:

```
do оператор; while (вираз);
```

Приклад 3. Цикл типу **`do – while`**.

Необхідно вгадати задане у програмі число. Один з варіантів реалізації:

```
…
do cin>>r;
while (r!=13);
cout<<"Ви вгадали число.";
…
```

В цьому прикладі користувач вводить числа до тих пір, поки не буде введено число 13. Після цього видається повідомлення про те, що ви вгадали.

Керуючі оператори в циклах

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

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 циклом оператора. Якщо оператор **`break`** знаходиться всередині вкладених циклів, то його дія поширюється тільки на той цикл, в якому він безпосередньо знаходиться.

Приклад 4. Використання оператора **`break`**.

Треба вгадати число 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.

Приклад 5. Використання оператора **continue**.

Вводяться числа місяця для обробки. Необхідно здійснити перевірку коректності вводу. Число 31 буде кінцем обробки.

```
…
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) в мові С++ є поганим засобом. Його використання 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 нуль, 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 **{}**.

Приклад 6. Цикл типу **for**.

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

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

Цикл for по діапазону

Цикл типу for по діапазону (range based) використовується для контейнерних типів даних, у випадку коли потрібно ітерувати по елементах контейнера (наприклад, масива або вектора).

Цикл по діапазону має наступну форму запису:

```
for(оголошення елементу : масив)
```

Оскільки оголошений елемент має бути того ж типу що і елементи масиву, часто використовують ключове слово auto (автоматичне визначення типу).

Приклад. Цикл типу **по діапазону.**

```
int array[7] = { 10, 7, 5, 4, 3, 1 };
for (auto i : array){
    std::cout << i << “, ”;
}
// вивід 10, 7, 5, 4, 3, 1,
```

Вкладені цикли

Вкладеним циклом називають конструкцію, в якій один цикл виконується всередині другого. Внутрішній цикл виконується повністю під час кожної ітерації зовнішнього циклу.

Приклад 7. Вкладені цикли.

Треба заповнити увесь екран символами **'#'**. Можливий варіант розв'язання має вигляд:

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

В цій програмі 25 разів виконується виведення по 80 символів.

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

Приклад 8. Ввести десять значень днів місяця з перевіркою правильності вводу.

```
…
for ( 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.

Приклади реалізації циклів

Виникає питання: 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 циклічного блоку програми. 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:

Приклад 9. Потрібно обчислити y5. Можливий варіант розв’язання має вигляд:

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

…

Приклад 10. Приріст при підрахунку, не рівний 1.

…

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

…

Приклад 11. Використання символів в якості лічильника.

Треба надрукувати алфавіт. Можливе рішення має вигляд:

…

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

…

Приклад 12. Зміна лічильника в геометричній прогресії. Треба підрахувати борг. Можливе рішення має вигляд:

…

```
for ( k=100;  k<185;  k*=1.1)  cout<<"Борг="<<k;
```

…

Приклад 13. Використання в якості лічильника виразу.

…

```
for ( k=1; z<=196; z=5*k+23 ) cout<<z;
```

…

Приклад 14. Неповний список виразів в заголовку тіла циклу.

…

```
for  (p=2; p<=202;)  p=p+n/k;
```

…

Приклад 15. Будь-який перший вираз в заголовку циклу.

…

```
for ( cout<<"Введіть числа."; p<=30;) cin>>p;
```

…

Приклад 16. Зміна управляючих змінних в тілі циклу.

…

```
delta=0.1;
for (k=1; k<500; k+=delta)
if (a>b) delta=0.5;
```

…

Приклад 17. Використання операції «кома» в специфікації циклу.

…

```
for ( i=1, r=1; i<=10; i++, r*=y )
cout<<"y в степені "<<i<<" дорівнює:"<<r;
```

…

2.2. Приклади

Приклад 1

Використання оператора **`switch`**.

Проаналізуємо значення змінної **rez**, яка є отриманою оцінкою.

```
…
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

Порівняти два значення змінних і вивести на екран значення більшої змінної, причому 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 3

Програма для підрахунку середньої оцінки учня.

```
#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 4

За допомогою циклу **for** вивести 5 степенів 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

2.3.1. Проаналізувати умову задачі.

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

2.3.3. Результати роботи оформити протоколом.

2.4. Варіанти завдань

Частина 1

Розв'язати наступні задачі, використовуючи умовну операцію:

1. Створіть 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. Створіть текстове меню, в якому при виборі першого пункту обчислюється значення квадрату введеного числа, при виборі другого пункту – значення кубу, при виборі третього – четвертого степеня числа.
2. Створіть 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. Створіть 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*2*3*…*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 $f_0=f_1=1$; $f_n=f_{n-1}+f_{n-2}$ 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 $f_0=f_1=1$; $f_n=f_{n-1}+f_{n-2}$ 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.

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 An=(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 a1+a2-a3+a4-a5+...+an, де n – кількість елементів послідовності.

2.5. Контрольні запитання і завдання

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?

Лабораторна робота №3
Реалізація функцій

Ключові слова: Поняття функцій, прототип, заголовок, реалізація, тіло функції, формальні та фактичні параметри, вбудовані та перевантажені функції, передача параметрів за замовчуванням, рекурсивні функції, стек, область видимості та класи пам'яті.

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 створення точки входу в об'єктний модуль.
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.
Із поняттям функції у мові C++ пов'язано три наступних компоненти:
— 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 повертається. У випадку 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 являє собою 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 викликаючу функцію і може використовуватися для передачі обчисленого значення функції. У тілі функції може бути декілька операторів **return**, але

може не бути й жодного. В останньому випадку повернення у викликаючу програму відбувається 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 формою такий же, як і заголовок функції, і 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.

При наявності прототипу функції, які викликаються, не зобов'язані розміщатися в одному файлі з функцією, що їх викликає.

Виклик функції може бути оформлений у вигляді оператора, якщо у функції відсутнє значення, що повертається, або у вигляді виразу, якщо існує значення, що повертається.

У першому випадку оператор має наступний формат:

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

Наприклад, `f(x);`.

В другому випадку вираз записується у такий спосіб: `h=f(x);`.

Значення обчисленого виразу є значенням функції, що повертається. Значення, що повертається, передається в місце виклику функції і є результатом її роботи.

Число і типи формальних аргументів повинні співпадати з числом і типом фактичних параметрів функції. При виклику функції фактичні параметри підставляються замість формальних аргументів. Після виклику функції значення аргументу, яке відповідає формальному параметру, використовується в тілі функції, що викликається. Такий спосіб виклику функцій називається *виклик по значенню*: змінні передаються функції в якості аргументів, їх значення копіюються у відповідні локальні змінні функції (формальні параметри), а самі фактичні змінні в основній програмі не змінюються [**Ошибка! Источник ссылки не найден.**, **Ошибка! Источник ссылки не найден.**].

Області дії змінних

При оголошенні змінних у програмі велике значення має те місце, де вона оголошена. Від того, де оголошена змінна, залежить можливість її використання. У C++ можливі три місця оголошення змінних.

По-перше, поза будь-яких функцій, у тому числі і **`main()`**. Така змінна називається *глобальною* і може використовуватися в будь-якому місці програми від місця оголошення і до кінця програми.

По-друге, змінна може бути оголошена всередині блоку, у тому числі й всередині тіла функції. Оголошена в такий спосіб змінна називається

локальною і може використовуватися лише всередині блоку. Така змінна поза блоком, в якому вона оголошена, недоступна.
По-третє, змінна може бути оголошена в якості параметра функції. Крім спеціального призначення, а саме для передачі даних у функцію, параметр можна розглядати як локальну змінну для тіла функції.
Приклад. Скласти програму для обчислення суми `k` чисел.

```
#include <iostream.h>

void sum(int);// прототип функції
int s = 0;// глобальна змінна

void main()
{
int i, b, k;// локальні змінні
cout << "\nВведіть кількість доданків";
cin >> k;
for(i = 0; i < k; i++)
{
cout << "\nВведіть новий доданок";
cin >> b;
sum(b);// виклик функції
}
cout << "\ns = "<< s;
}

void sum(int c)
{
s = s + c;
}
```

У цій програмі змінна **`s`** є глобальною, вона доступна з обох функцій програми — **`main()`** та **`sum()`**, а змінні **`i`**, **`b`**, **`k`** та **`c`** — локальні, доступні тільки у тих функціях, де вони оголошені.
Якщо глобальна і локальна змінні мають одне і теж ім’я, тоді вважається, що оголошені дві різні змінні зі своїми областями використання. При цьому локальна змінна буде видима у тій функції, де вона оголошена, а глобальна у всій програмі за виключенням функції, у якій оголошена локальна змінна.

Передача аргументів за замовчуванням

Значення формальних параметрів можуть бути задані за замовчуванням. Зазвичай це константа, яка часто зустрічається при виклику функції. При цьому при описанні функції перелік параметрів за замовчуванням повинен міститися в кінці списку формальних змінних функції. Розглянемо наступні приклади [**Ошибка! Источник ссылки не найден.**]:

```
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 назви функції спрямований відобразити її основне призначення. Перевантаження використовує одну й ту саму назву для декількох операторів або функцій. Вибір варіанту залежить від типу аргументів оператора або функції. Наприклад:

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

Вбудовані функції в C++ забезпечується ключовим словом `inline`. Воно розташовується перед оголошенням функції, коли необхідно, щоб код в тілі функції вбудовувався в місце виклику функції [**Ошибка! Источник ссылки не найден.**].

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

Обмеження компілятора не дозволяє вбудовувати складні функції.

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 являє собою функцію, у тілі якої здійснюється виклик цієї ж функції.

Нехай необхідно створити програму 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 << "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 факторіал нуля дорівнює 1 за означенням, то в тілі функції передбачений і цей варіант. У випадку коли аргумент функції `fact()` відмінний від 0 та 1, викликаємо функцію `fact()` із зменшеним на одиницю значенням параметра і множимо результат на значення поточного параметра. Таким чином, в результаті вбудованих викликів функцій отримаємо наступний результат:

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

Класи пам'яті

Кожна змінна і функція в C++ має два атрибути:
— тип;
— клас пам'яті.
Існує чотири класи пам'яті:
— автоматичний – `auto`;
— зовнішній – `extern`;
— регістровий – `register`;
— статичний – `static`.

Автоматичний. Змінна, оголошена всередині функції, за замовчуванням є автоматичною. Оголошення змінних всередині блоку неявно отримують автоматичний клас пам'яті.
Приклад (явного використання)

```
auto int a, b, c;
auto float d = 5.38;
```

Система виділяє пам'ять для автоматичних змінних при вході в блок в стеку.
Зовнішній. Один із способів передачі інформації між блоками та функціями полягає у використанні зовнішніх змінних. Якщо змінна оголошена ззовні

функції – її клас пам'яті `extern`. Розглянемо наступний приклад [**Ошибка! Источник ссылки не найден.**].

Файл `circle.cpp`.

```
const double pi = 3.14159; //локальна до файлу circle.cpp змінна

double circle(double radius)
{
   return (pi * radius * radius);
}
```

Файл `cir_main.cpp`.

```
#include <iostream.h>

double circle(double); //функція автоматично є зовнішньою

int main()
{
   double x = 3.5;
   cout << circle(x) << "is area of circle of radius "
<< x;
}
```

Регістровий. При використанні регістрового класу пам'яті відповідні змінні будуть розміщено у швидких регістрах пам'яті (при наявності такої можливості). Якщо компілятор не може виділити фізичний регістр, клас пам'яті стає автоматичним [**Ошибка! Источник ссылки не найден.**].

```
for (register i = 0; i < LIMIT; i++)
{
   …
}
```

Оголошення `register i = 0;` рівнозначно `register int i = 0;`.

Статичний. Змінні, оголошені як `static`, мають два важливих застосування [**Ошибка! Источник ссылки не найден.**]:

— локальна змінна зберігає значення при повторному вході в блок;

— забезпечення механізму закритості (privacy).

Наведемо приклад зберігання значення статичної змінної [**Ошибка! Источник ссылки не найден.**].

```
#include <iostream.h>

int count()
{
```

```
   static int called = 0; // статична змінна, яка
зберігає своє значення при кожному виклику функції
count()
   ++called;
   return called;
}

main()
{
   int call_times;
   for (int i = 0; i < 5; ++i)
      call_times = count();
   cout << "\nThe function is called " << call_times <<
" times\n";
}
```

Наведемо приклад забезпечення закритості [**Ошибка! Источник ссылки не найден.**].

```
static int static_func(int a)
{
   ...
}

int non_static_func(int a)
{
   ...
   b = static_func(a); //доступно лише в цьому файлі; в
інших – не доступно
   ...
}
```

3.2. Приклад

Розглянемо приклад використання функції, яка обчислює максимум 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;
}
```

3.3. Порядок виконання роботи

5.3.1. Проаналізувати умову задачі.
5.3.2. Розробити алгоритм та створити програму розв'язання задачі згідно з номером варіанту.
5.3.3. Результати роботи оформити протоколом.

3.4. Варіанти завдань

Написати функцію, яка обчислює значення виразу, та визначити її значення для введених користувачем значень фактичних параметрів. При цьому знайти область визначення функцій та забезпечити необхідну реакцію програми на некоректні ситуації.

Написати функцію, яка обчислює значення виразу, та визначити її значення для введених користувачем значень фактичних параметрів. При цьому знайти область визначення функцій та забезпечити необхідну реакцію програми на некоректні ситуації.

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

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

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

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

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

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

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

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

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) $y=\sqrt{\dfrac{a-2}{2a+b}+\dfrac{\sin3a}{\sqrt{\cos b}}}$;

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

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

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

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

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

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

Завдання підвищеної складності.

1. Дано натуральне 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:

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

3. Для заданих x та n обчислити значення виразу:

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

4. Для заданого n обчислити значення виразу

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

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

3.5. 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?

Лабораторна робота №4
Робота з пам'яттю

Ключові слова: області пам'яті, доступні програмам на C++ та їх призначення; засоби C++ для роботи з пам'яттю; вказівники та їх природа; адресна арифметика; створення динамічних масивів.

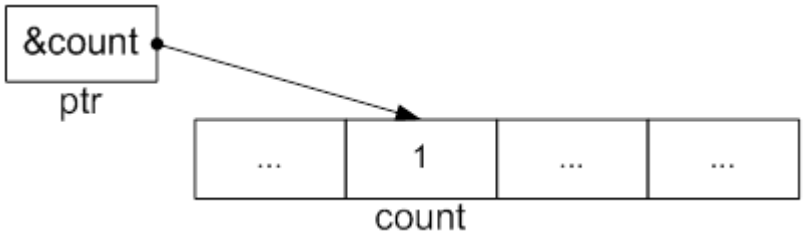
Мета роботи: отримати навички роботи з вказівниками, посиланнями та динамічними масивами.

4.1. Теоретичні відомості

Вказівники

Вказівник — це тип даних, який використовується для зберігання адрес змінних і об'єктів. Для даного типу `T`, тип `T*` є «вказівником на `T`» [4]. Тобто змінна типу `T*` містить адресу об'єкта типу `T`. Наприклад:

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



В мові C++ наявна операція визначення адреси — **`&`**, за допомогою якої визначається адреса комірки пам'яті, де зберігається значення заданої змінної. Наприклад, якщо **`vr`** — ім'я змінної, то **`&vr`** — адреса цієї змінної. Нехай змінна типу вказівник має ім'я **`ptr`**. Тоді в якості значення їй можна присвоїти адресу за допомогою наступного оператора:

```
ptr = &vr;
```

В 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 направо. Цю операцію не слід плутати з бінарною операцією множення. Нехай **`ptr`** — вказівник, тоді **`*ptr`** — це значення змінної, на яку вказує **`ptr`**. Наприклад [3]:

```
int a = 1, b;
int* ptr = &a; //містить адресу змінної a
```

```
    cout << " Змінна a = " << (*ptr) << "зберігається за
адресою " << ptr;

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

Посилання

Посилання (reference) являє собою видозмінену форму вказівника, яка використовується в якості *псевдоніму* (другого імені) змінної. У зв'язку з цим посилання не потребують додаткової пам'яті. Для визначення посилання використовують символ **&** (амперсант), який ставиться перед змінною-посиланням.

Змінні типу посилання можуть використовуватися в наступних цілях:

— замість передачі у функцію об'єкта за значенням;

— для визначення конструктора копії при реалізації класів;

— для перевантаження унарних операцій.

Приклад. Використання посилань.

```
#include <iostream.h>

int main()
{
   int t = 13,
   int &ref = t;// ініціалізація посилання на t
                // тепер ref є синонімом 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;
}
```

В даному випадку ми використовували посилання `ref` в якості псевдоніму змінної `t`. В цій ситуації воно називається *незалежним посиланням* (independent reference) і повинно бути ініціалізоване під час оголошення. Такий спосіб використання посилань може призвести до фатальних помилок, які важко виявити через виникнення плутанини у використанні змінних.

Способи передачі аргументів в функцію: за вказівником та посиланням

В лабораторній роботі № 3 було розглянуто один спосіб передачі аргументів в функції, а саме за значенням. Однак такий спосіб має наступні недоліки: передача об'єктів великого об'єму вимагатиме великих витрат часу та пам'яті; іноді необхідно модифікувати значення аргументів в тілі функції. Для цього використовується передача аргументів за *вказівником* (або *адресою*) та *посиланням*.

Під час передачі аргументів за вказівником в стек заноситься копія не значення, а копія вказівника. Відповідно функція оперує з адресами на комірки пам'яті, де знаходиться значення аргументу функції, і може його змінювати.

Приклад. Функція з параметром-вказівником.

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

void sqr(int*);// прототип функції

int main()
{
   int t = 3;
   cout << "Початкове значення t:" << t; // виводить
3
   sqr(&t); // передача адреси змінної t
   cout<<"\nОстаточне значення t:" << t; // виводить
9
   return 0;
}

void sqr(int* x)
{
   (*x)*= (*x);      //      використано      операцію
розіменування
}
```

Третій спосіб передачі аргументів — за *посиланням*. В цьому випадку параметри функції передаються за допомогою посилання. Тобто, в тіло функції передається адреса параметра і всі звернення до параметра неявно розіменуються. Фактично ми створюємо псевдоніми аргументів функції, за допомогою яких можемо змінювати вхідні значення параметрів. Використання посилань замість вказівників звільняє від необхідності застосовувати операції копіювання адрес та розіменування.

Приклад. Функція з параметром-посилання.

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

void sqr(int &);// прототип функції
```

```
int main()
{
   int t = 3;
   cout << "Початкове значення t:" << t; // виводить
3
   sqr(t);
   cout<<"\nОстаточне значення t:" << t; // виводить
9
   return 0;
}

void sqr(int& x)
{
   x*= x;
}
```

Масиви та вказівники

Масив являє собою деяку послідовність значень одного типу. Оголошення стандартного (статичного) масиву виділяє необхідну пам'ять (кількість елементів помножене на об'єм пам'яті, який займає відповідний тип даних), починаючи з базової адреси. Взагалі кажучи, ім'я масиву являє собою *постійний вказівник*, який ініціалізовано цією базовою адресою [3]. 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 постійний вказівник з фіксованою (базовою) адресою [3].

Розглянемо наступний приклад.

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

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 [3]:

```
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 [3].

```
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:

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]	...

Зауважимо, 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 — це область пам'ять, яка надається

системою, для об'єктів, час життя яких напряму керується програмістом [3]. 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. Загальний формат оператора new:

```
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;
```

Приклад. 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.

```
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. Оператор int* mas = new int[n] виконує дві дії: оголошується змінна типу вказівник 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 типу 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.

Дуже часто при програмуванні виникає необхідність створення багатовимірних динамічних об'єктів. Програмісти-початківці за аналогією з

поданим способом створення одновимірних динамічних масивів для двовимірного динамічного масиву розмірності `n*k` запишуть наступне

```
int* mas = new int[n][k]; // Невірно! Помилка!
```

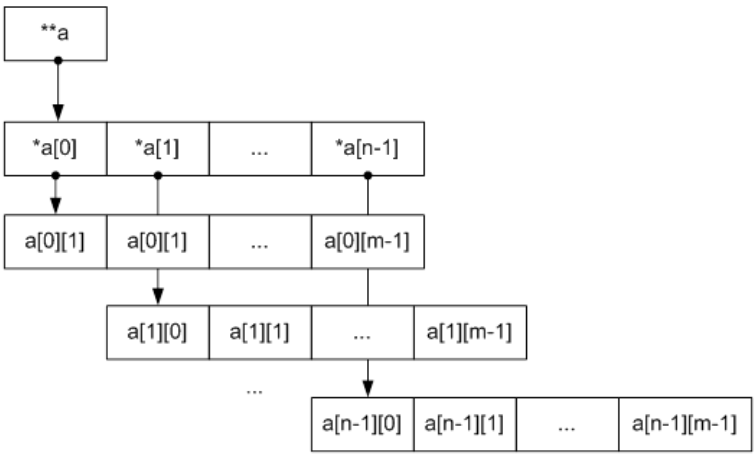
Такий спосіб виділення пам'яті не дасть коректного результату. Наведемо приклад створення динамічного двовимірного масиву.

```
#include <iostream.h>

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 ми створюємо подвійний вказівник `int** a`: вказівник на масив вказівників. Під цей масив вказівників ми виділяємо пам'ять 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*m.



Інший спосіб створення багатовимірного масиву полягає у використанні одновимірного. Розглянемо приклад реалізації матриці на основі одновимірного динамічного масиву.

```
#include <iostream.h>

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

   int* a; //a - вказівник
   a = new int[n*m]; //виділення пам'яті для масиву розмірності n*m
...
// Вивід елементів масиву
   for(int i = 0; i < n; i++){
      for(int j = 0; j < m; j++){
         cout << a[i*m + j] << "   ";
      }
      cout<<endl;
   }

// Видалення пам'яті
   delete[] a;
   return 0;
}
```

	0	1		m-1
0	a[0]	a[1]	...	a[m-1]
1	a[m]	a[m+1]	...	a[2*m-1]
	...	...	...	...
n-1	a[(n-1)*m]	a[(n-1)*m +1]	...	a[n*m-1]

Передача масивів у функції

Якщо в якості аргументу функції використовується масив, тоді передається вказівник на його перший елемент [4]. Наприклад:

```
int strlen (const char*);

void f(){
   char v[] = "массив";
   int i = strlen (v);
   int j = strlen ("Nicholas");
}
```

Тобто при виклику функції аргументи типу `T[]` будуть перетворено в тип `T*`. Фактично ми будемо передавати вказівники на елементи масиву, і через цей вказівник зможемо змінювати значення елементів масиву в тілі функції. Іншими словами, масиви не можна передавати по значенню на відміну від інших типів даних [4]. Розмір масиву невідомий в тілі функції і це може створювати труднощі. Для її розв'язання можна скористатися наступними рішеннями. Рядки символів є обмеженими і їх кількість можна обчислити. В інших випадках розмір масиву можна передавати в якості іншого аргументу функції. Наприклад:

```
void f(int* mas, int mas_size);
```

Але з багатовимірними масиви виникають труднощі. Розглянемо функцію, яка оперує с двовимірним масивом (матрицею). Якщо розміри відомі під час компіляції, проблем з цим не виникає:

```
void print(int mas[3][5]){
   for (int i = 0; i < 3; i ++){
      for (int j = 0; j < 5; j++)
         cout << mas[i][j] << '\t';
cout << '\n';
   }
}
```

Матриця (багатовимірний масив) передається як вказівник (а не копіюється). Перша розмірність масиву не впливає на проблему знаходження положення елемента; вона просто вказує, скільки є елементів (в даному

випадку 3) відповідного типа (тут `int[5]`). Знаючи тільки другу розмірність 5, ми зможемо знайти положення `mas[i][5]` для будь-якого значення `i`. Тому першу розмірність можна передати в якості аргументу:

```
void print (int mas[][5], int diml){
   for (int i = 0; i < diml; i++){
      for (int j = 0; j < 5; j++)
         cout << mas[i][j] << '\t';
      cout << '\n';
   }
}
```

Однак простим чином передати дві розмірності в якості аргументів вже не виходить. Наступне рішення буде некоректним:

```
void print(int mas[][], int diml, int dim2){
   for (int i = 0; i < diml; i++){
      for (int j = 0; j < dim2; j++)
         cout << mas[i][j] << '\t';
      cout << '\n';
   }
}
```

По-перше, оголошення `mas[][]` некоректне, оскільки для того щоб знайти елемент, необхідно знати другу розмірність двовимірного масиву (`[i*dim2 + j]`). По-друге, вираз `mas[i][j]` буде інтерпретуватися як `*(*(m+i)+j`, і це не те, що ми очікуємо від нашої програми. Правильне рішення полягає у використанні вказівника:

```
void print(int mas[][], int diml, int dim2){
   for (int i = 0; i < diml; i++){
      for (int j = 0; j < dim2; j++)
         cout << mas[i*dim2 + j] << '\t';
      cout << '\n';
   }
}
```

Для виклику цієї функції ми передаємо вказівник:

```
int v[3][5] =
   {{0,  1,  2,  3,  4},
   {10, 11, 12, 13, 14},
   {20, 21, 22, 23, 24}};

print(&v[0][0], 3, 5);
// або print(v[0], 3, 5);
```

Вказівники на функцію

З функціями можна роботи дві речі: викликати та отримати її адресу. Вказівник, отриманий шляхом взятті адреси функції, можна використовувати для її виклику [4]. Наприклад:

```
void error (string s) {/*... */}
void (*efct) (string); // вказівник на функцію

void f()
{
   efct = &error; // efct вказує на функцію error
   efct("помилка"); // виклик функції error через вказівник efct
}
```

Компілятор розпізнає, що `efct` є вказівником та викликає функцію, на яку він вказує. Тобто, розіменування вказівника на функцію за допомогою оператора `*` не є обов'язковим. Аналогічно, необов'язково використовувати `&` для отримання адреси функції:

Аргументи вказівників на функцію оголошуються таким самим чином, як і аргументи самих функцій. При присвоюванні типи функцій повинні точно співпадати. Наприклад [4]:

```
void (*pf) (string); // вказівник на void(string)
void f1 (string); // void(string)
int f2 (string); // int(string)
void f3 (int*); // void(int*)

void f*(){
   pf = &fl; // коректно
   pf = &f2; // помилка: не той повертає мий тип
   pf = &f3; // помилка: не той тип аргументу
   pf("НТУУ КПІ"); //коректно
   pf(0); // помилка: не той тип аргументу
   int i = pf("НТУУ КПІ"); // помилка: присвоювання void змінній int
}
```

Правила передачі аргументів при виклику функцій через вказівники ті самі, що і при безпосередньому виклику функцій.

Дуже часто корисно використовувати масив вказівників на функції. Це може бути, наприклад, система меню в редакторі. Вона може бути реалізована з використанням масиву вказівників на функції, які виконують відповідні операції. Розглянемо загальну ідею.

```
typedef void (*PF) ();

PF edit_ops[] = { // команди редагування
   &cut, &paste, &copy, search };
PF file_ops[] = { // файлові операції
```

```
    &open, &append, &close, &write };
```

Тепер ми можемо визначити та ініціалізувати вказівники для дій, які вибираються із меню и зв'язаних з ними кнопками:

```
PF* button2 = edit_ops;
PF* button3 = file_ops;
```

Тобто при натисканні кнопки `button2` будуть ініціалізовані команди редагування тексту, а `button3` — файлові операції. Коли користувач вибере пункт меню, наприклад пункт 3, з нажатою кнопкою 2, буде виконана наступна операція (виклик відповідної функції):

```
button2[2](); // виклик третьої функції button2, тобто copy()
```

Засоби аналізу продуктивності програм

При написанні складних програмних систем часто виникає задача оптимізації її функціонування, наприклад, за часом виконання та використаною пам'яттю. Для того, щоб визначити «вузькі місця» програми, що розробляється, необхідно використовувати спеціальні засоби, за допомогою яких можна проаналізувати її продуктивність. Такі засоби називаються *профайлерами* (profiling programs).

Одним із популярних профайлерів є програмний продукт Intel® VTune™ Performance Analyzer. Даний засіб не залежить від компілятора та може застосовуватися для програм, які написано на різних мовах програмування, наприклад, C, C++, Fortran, C#, Java, тощо. Варто відзначити, що існують версії цього продукту для різних операційних систем, зокрема Windows та Linux.

4.2. Приклади

Використовуючи вказівники, вивести на екран масив із заданою кількістю елементів.

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

int main()
{
    int *a;
    int n;
    cout << "Enter number of elements in your array:" << endl;
    cin >> n;

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

```
    {
        *(a + i) = i + 1;
        cout << *(a+i) <<endl;
    }
    delete []a;
    return 0;
}
```

4.3. Порядок виконання роботи

4.3.1. Проаналізувати умову задачі.

4.3.2. Розробити алгоритм та створити програму розв'язання задачі згідно з номером варіанту.

4.3.3. Результати роботи оформити протоколом.

4.4. Варіанти завдань

Реалізувати наступні завдання з використанням динамічних масивів. Для доступу до елементів 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:

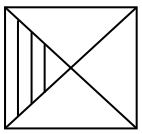
1	3	4	10	11
2	5	9	12	19
6	8	13	18	20
7	14	17	21	24
15	16	22	23	25

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.

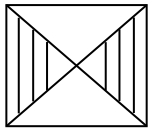
16. Заповнити двовимірний масив розмірності *n*x*n* одиницями в шаховому порядку, починаючи з верхнього лівого кута.

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.

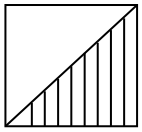
19. Дано квадратну матрицю розмірності *n*x*n*. Знайти суму елементів та максимальний елемент у заштрихованій області.



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.



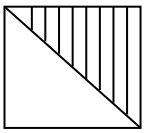
23. Заповнити двовимірний масив розмірності *n*x*m* з клавіатури лише числами кратними 3, передбачивши захист елементів цього масиву від неправильного введення. Знайти суму тих елементів масиву, які без остачі діляться на 9.

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.

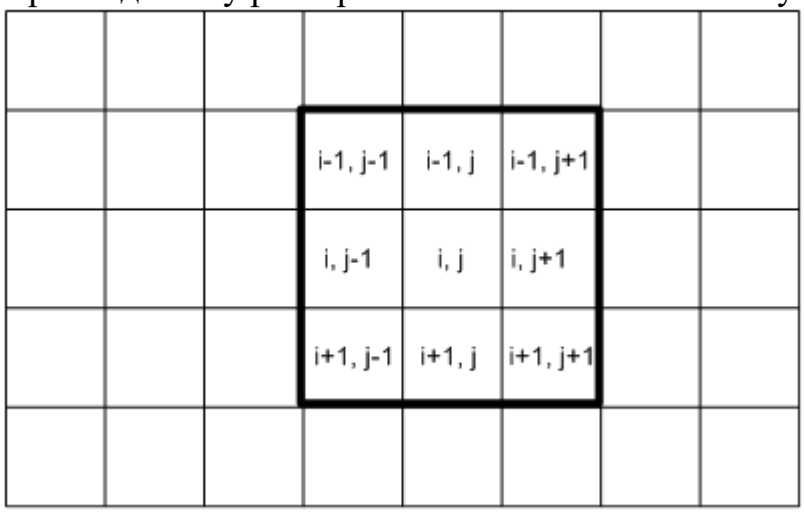
29. Побудувати і вивести на екран "латинський" квадрат - масив, що складається з n різних чисел, всіх по n раз розташованих так, що в кожному рядку й стовпці кожне число зустрічається лише один раз.

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]` градієнт 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.

4.5. 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 сторження багатовимірних динамічних масивів Ви знаєте?
12. Які способи передачі масивів у функції Ви знаєте? В яких випадках виникають некоректності і як їх розв’язувати?

13. Що таке вказівник на функцію? Яким чином можна викликати функції через вказівник? Наведіть області застосування вказівників на функцію.

Лабораторна робота №5. Робота з файлами

Мета роботи: Отримати навички роботи з файлами з використанням засобів моіи C++.

5.1 Теоретичні відомості

У мові C/C++ 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 C/C++ підтримує потокове введення/виведення (заголовний файл fstream) .

Мова C є фундаментом C++. При цьому C++ підтримує всі засоби роботи з файлами мови C. Тому при використанні C-коду в програмі на мові C++ немає необхідності змінювати процедури введення/виведення. В той же час слід зазначити, що при написанні програм на C++ зазвичай більш зручно використовувати саме засоби C++. З точки зору засобів програмування C/C++ файл є іменованим місцем на жорсткому диску, що у загальному випадку може бути пов'язаний з реальним фізичним пристроєм (рис. 1).

Рис. 1. Використання файлів в програмі C/C++

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.

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.

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.

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:

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 оператора 'або' (|). Наприклад, режим ios::binary | ios::in визначає доступ тільки для читання бінарного файлу.

Приклад.

```
// 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.

// 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();
```

5.2 Приклад

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;
}
```

5.3. Порядок виконання роботи

5.3.1. Проаналізувати умову задачі.

5.3.2. Розробити алгоритм та створити програму розв’язання задачі згідно з номером варіанту.

5.3.3. Результати роботи оформити протоколом.

5.4 Варіанти завдань

Початковий рівень

1. В файлі test.in записано 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 NxN. Знайдіть її детермінант. Результат запишіть в test.out. Для роботи з файлами використовуйте функції cstdio.
4. В файлі test.in записано матрицю NxN. 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.

Середній рівень

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 символі

18.в в бінарному файлі. Для роботи з файлами використовуйте функції cstdio.

19.Створіть програму, що буде знаходити всі текстові строки довжиною більше 5 символів в бінарному файлі. Для роботи з файлами використовуйте функції fstream.

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 cstdio.

21.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.

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 cstdio.

23.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.

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 cstdio.

25.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.
2. Створіть програму, що підраховує частоти входження слів в текстовому файлі.
3. Створіть програму, що підраховує частоти N-грамм (N > 2) у бінарному файлі.

5.5 Контрольні питання

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

Лабораторна робота № 6
Ознайомлення з бібліотекою шаблонів

Ключові слова: бібліотека стандартних шаблонів, STL, контейнери, простір імен, ітератори.

Мета роботи: отримати навички роботи з бібліотекою STL 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 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;
}

void main()
{
        using namespace spaceA;
        cout << MyVal << " " << "spaceA" << endl;
        Test();
        cout << spaceC::MyVal << " " << "spaceC" << endl;
}
```

Структура бібліотеки STL

STL — це бібліотека стандартних шаблонів (Standard Template Library). Вона містить часто використовувані структури даних (або контейнери), наприклад, динамічні масиви, двонаправлені списки, стеки, черги тощо. Крім цього, STL містить множину алгоритмів сортування (як на всій, так і на частині множини), пошуку мінімального та максимального значень, тощо. Кожний алгоритм дозволяє працювати з різними типами контейнерів. Іншими словами, один і той же алгоритм сортування можна використовувати як для динамічного масиву, так і для стеку.

Таким чином, у складі бібліотеки STL можна виділити три різних підсистеми.

1. **Контейнери**. Частину контейнерів складають часто вживані прості структури даних, такі як динамічні масиви, списки, черги та ін. Інша частина контейнерів складається з асоціативних контейнерів. Основною рисою асоціативних контейнерів є забезпечення пошуку значень за ключами. При цьому ключ може бути будь-яким. Аналогію таких асоціативних контейнерів легко знайти у реальному житті. Саме за ключами (прізвищем власника або назвою організації) можна шукати номери телефонів у телефонному довіднику.

Слід зазначити, що у кожному контейнері крім просто даних можна використовувати також методи для роботи із цими даними (для додавання, пошуку, видалення й ін.), оскільки кожний контейнер по суті є класом.
Більш докладно класи розглядаються у курсі «Програмування. Об'єктно-орієнтований підхід».
2. **Алгоритми**. Слід зазначити, що алгоритми не є частиною контейнерів, а утворюють окрему підсистему. Але при цьому майже будь-який алгоритм може застосовуватися до майже будь-якого контейнера. Іншими словами, при використанні деякого алгоритму для обробки деякого контейнера, цей контейнер передається в якості параметра.
3. **Ітератори**. Ітератор — це деякий покажчик, що дозволяє перебрати всі елементи контейнера. Ітератори відіграють таку ж роль, що й індекс елемента звичайного масиву. За допомогою індексу масиву можна отримати деякий елемент масиву, а за допомогою ітератора — деякий елемент контейнера. Ітератори бувають різних типів: для переміщення у контейнері тільки вперед, в обидва боки тощо.

Контейнери STL C++

Розглянемо приклад використання контейнерів на прикладі векторів. Вектор має значну перевагу в порівнянні з масивом, оскільки його розмірність автоматично збільшується при додаванні нових елементів. Щоб забезпечити можливість використання векторів, необхідно включити у свою програму заголовний файл **<vector>** і використовувати простір імен **std**.

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

Щоб створити цілочисельний вектор, необхідно оголосити його :

```
vector<int> i_vec;
```

Тут оголошена змінна **i_vec**, що є порожнім цілочисельним вектором (на це вказує **<int>**). Для додавання елементів до цього вектора використовується метод **push_back()**:

```
i_vec.push_back(3); // поміщаємо 3 у кінець вектора
i_vec.push_back(1); // потім добавляємо 1
i_vec.push_back(2); // і, нарешті, 2
// тепер i_vec містить три елементи
// виведення вектора
for (int ix = 0; ix < i_vec.size(); ix++)
cout << i_vec[ix] << " ";
// виводиться: 3 1 2
```

Подібно іншим типам, векторам можна присвоювати значення з використанням операції = і виконувати над ними операцію порівняння ==. Наприклад:

```
vector<int> vecA, vecB;
vecA.push_back(3);  vecA.push_back(2);
vecB = vecA; // vecB = (3, 2)
vecB[1] = 0; // vecB = (3, 0)
```

```
if (vecA == vecB) cout << "Рівні!" << endl;
```

Операція == виконує поелементне порівняння двох векторів. У даному випадку вектори не рівні між собою.

Приклад 2. Використання шаблону **vector** як динамічного одновимірного масиву.

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

void main()
{
   // створення вектору з цілих чисел
   vector <int> k;

   // додавання елементу в кінець вектора
   k.push_back(22);
   k.push_back(11);
   k.push_back(4);

   // вивід у поток cout всіх елементів вектора
   for (int i = 0; i < k.size(); i++)
   {
      cout << k[i] << "\n";
   }
   cout << "***\n";

   // видалення елементу з кінця вектора
   k.pop_back();

   // вивід всіх елементів вектора
   for (i = 0; i < k.size(); i++)
   {
      cout << k[i] << "\n";
   }
   cout << "***\n";

   // видалення всіх елементів вектора
   k.clear();

   // вектор порожній ?
```

```
    if (k.empty) {
        cout << "Вектор порожній" << endl;
    }
}
```

Слід зазначити, що використані функції та змінні шаблону **vector** (такі як **push_back**, **pop_back**, **clear** і **empty**) є методами відповідного класу.
До складу бібліотеки STL входять також і інші класи-контейнери.

list	Двонаправлений список. Доступ до елементів можливий тільки послідовний, із двох боків списку.
stack	Стек.
map	Асоціативний список. Дозволяє зберігати дані у вигляді пари «ключ-значення» (з кожним ключем пов'язане тільки одне значення)
multimap	Асоціативний список. Дозволяє зберігати дані у вигляді пари «ключ-значення» (з ключем може бути пов'язано більше одного значення).
set	Множина, усі якої елементи унікальні.
multiset	Така множина дозволяє зберігати повторювані, або однакові, елементи.
queue	Черга.
Deque	Двостороння черга.
Bitset	Набір бітів.

Для використання цих шаблонів ви повинні на початку програми включити необхідні заготовочні файли за допомогою директиви **include**. У більшості випадків імена заготовочних файлів співпадають з іменем шаблону (так, наприклад, для використання шаблону **list** потрібно написати **#include <list>**). Виключення становлять **multimap** (потребує **#include <map>**) і для **multiset** (потребує **#include <set>**).

Ітератори

У першому наближенні літератор — це покажчик на деякий елемент у контейнері. І, як і у випадку з покажчиками, отримати доступ до елемента контейнера можна шляхом розіменування літератора. Наприклад,

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

void main()
{
    // Оголошуємо вектор із цілих.
    vector <int> k;
```

```
    // Додаємо елементи в кінець вектора.
    k.push_back(22);
    k.push_back(11);
    k.push_back(4);
    // Оголошуємо ітератор.
    vector <int>::iterator p;
    // Встановлюємо ітератор на початок вектора
    // і пересуваємо його в циклі на наступну позицію
    for (p = k.begin(); p < k.end(); p++)
    {
       // Виводимо вміст елементів вектора
       // за допомогою розіменований ітератор
       cout << *p << "\n";
    }
}
```

У результаті роботи програми буде отримано числа 22, 11 і 4.

Алгоритми

Алгоритми — третя складова частина бібліотеки STL. У наведеному нижче прикладі алгоритм застосовано для сортування елементів вектора.

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

void main()
{
    // Оголошуємо вектор із цілих.
    vector <int> k;
    // Додаємо елементи в кінець вектора.
    k.push_back(22);
    k.push_back(11);
    k.push_back(4);
    k.push_back(100);
    vector <int>::iterator p;
    // Вивід невідсортованого вектора.
    for (p = k.begin(); p < k.end(); p++)
    {
       cout << *p << "\n";
    }
```

```
    // Сортування вектора.
    sort(k.begin(), k.end());

    // Вивід відсортованого вектора.
    cout << "sorted:\n";
    for (p = k.begin(); p<k.end(); p++)
    {
       cout << *p << "\n";
    }
}
```

При виклику функції **sort** (алгоритму сортування) у якості параметрів вказано діапазон елементів, які треба відсортувати. У наведеному прикладі вектор сортується повністю, з початку до кінця.
Результатом виконання програми буде вивід чисел 4, 11, 22, 100.
Слід зазначити, що застосований метод сортування (як і інші алгоритми) не належить шаблону vector, а входить до складу підсистеми алгоритмів.

Рядки в C++

В C++ з'явилася 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 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.

```
#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!”
```

Ось ще один приклад використання рядків.

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

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

6.2 Приклади

```
#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;
 }
```

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;
}

void main()
{
```

```
        using namespace spaceA;
        cout << MyVal << " " << "spaceA" << endl;
        Test();
        cout << spaceC::MyVal << " " << "spaceC" << endl;
}
```

Використання шаблону **vector** як динамічного одновимірного масиву.

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

void main()
{
   // створення вектору з цілих чисел
   vector <int> k;

   // додавання елементу в кінець вектора
   k.push_back(22);
   k.push_back(11);
   k.push_back(4);

   // вивід у поток cout всіх елементів вектора
   for (int i = 0; i < k.size(); i++)
   {
      cout << k[i] << "\n";
   }
   cout << "***\n";

   // видалення елементу з кінця вектора
   k.pop_back();

   // вивід всіх елементів вектора
   for (i = 0; i < k.size(); i++)
   {
      cout << k[i] << "\n";
   }
   cout << "***\n";

   // видалення всіх елементів вектора
   k.clear();
```

```
    // вектор порожній ?
    if (k.empty) {
        cout << "Вектор порожній" << endl;
    }
}
```

6.3. Порядок виконання роботи

6.2.1. Проаналізувати умову задачі.
6.2.2. Розробити алгоритми та створити програми розв’язання задач з розділів 10.4.1 та 10.4.2 згідно з номером варіанту.
6.2.3. Результати роботи оформити протоколом.

6.4. Варіанти завдань

Робота з рядками

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 "сссааддд"). Вивести отриманий рядок на екран.
20. Викреслити і-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?

Робота з рядками

6.3.2.1. Модифікувати програму, розроблену у лабораторній роботі № 4, з використанням шаблонів бібліотеки STL.
Завдання підвищеної складності
1.Реалізувати інформаційний довідник на основі класу

6.5 Контрольні запитання

1. Для чого призначена бібліотека STL C++?
2. Які підсистеми її складають?
3.Як у мові C++ можна працювати з символьними рядками?
4. Для чого використовується поняття простору імен?
5. Що таке ітератор та яких типів він буває?
6. Як пов’язані літератори та контейнери?

Лабораторна робота № 7
Структури

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

7.1. Теоретичні відомості

Структура - складений тип даних, в якому згруповані елементи різних типів.
Оголошення структурної змінної здійснюється в 2 етапи:

- оголошення шаблону структури як нового типу даних. На цьому етапі пам‘ять не виділяється, Формується тільки інформація про вміст структури;
- оголошення самої змінної. На цьому етапі виділяється пам‘ять для кожного поля (змінної), що описується в шаблоні структури.

У визначенні структури застосовується ключове слово **struct**, а сам формат визначення виглядає наступним чином:

```
struct ім'я_структури
{
     компоненти_структури
};
```

Як правило, розмір структури - це сума розмірів всіх її елементів. Визначити точно це можна за допомогою sizeof()

Приклад.
Шаблон структури, що описує книгу в бібліотеці.

```
struct Book
{
    char title[70];
    char author[50];
    int year;
    float price;
};
```

Після визначення структури її можна використовувати, визначивши об'єкт структури - змінну, яка буде представляти створений тип. Також після створення змінної структури можна звертатися до її елементів - отримувати їх значення або, навпаки, присвоювати їм нові значення. Для звернення до елементів структури використовується операція "точка":
Приклад коду

```
#include <iostream>
using namespace std;
struct Person
{
    char name[50];
    int age;
   };
int main()
{
    Person p1;
    cout << "Enter Full name: ";
    cin.get(p1.name, 50);
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Name: " << p1.name << endl;
    cout <<"Age: " << p1.age << endl;
    return 0;}
```

Масив структур

Змінні структур можуть бути елементами масивів, як змінні будь якого іншого типу.

Оголошення та введення масиву структур

Person p1[3];

```
    for(int i=0;i<3;i++)
    {
    cin.get(p1[i].name, 50);
    cin >> p1[i].age;
    }
```

7.2 Приклад

```
#include <iostream>
using namespace std;
struct Person
{
    char name[50];
    int age;
};

int main()
{
    Person p1[3];
    for (int i = 0; i < 3; i++)
```

```
    {
        cin.getline(p1[i].name, 50);
        cin >> p1[i].age;
        cin.get();
    }
    for (int i = 0; i < 3; i++)

    {
        cout << "Name: " << p1[i].name << endl;
        cout << "Age: " << p1[i].age << endl;
    }
    return 0;
}
```

Структури і функції

Великою перевагою використання структур, ніж окремих змінних, є можливість передати всю структуру в функцію, яка повинна працювати з її елементами

```
#include <iostream>
 struct Employee
{
    short id;
    int age;
    double salary;
};

void printInformation(Employee employee)
{
    std::cout << "ID: " << employee.id << "\n";
    std::cout << "Age: " << employee.age << "\n";
    std::cout << "Salary: " << employee.salary << "\n";
}
 int main()
{
    Employee john = { 21, 27, 28.45 };
    Employee james = { 22, 29, 19.29 };
     printInformation(john);
     std::cout << "\n";
      printInformation(james);
     return 0;
}
```

Вказівники на структури

Як і на будь-який інший тип, на структури можна вказати за власним типом покажчиків:

```
struct movies_t {
   заголовок рядка;
   int рік;
};
movies_t amovie;
movies_t * pmovie;
```

Вказівники на структури можна створювати і для безіменних структурних типів:

```
struct
{
    int age;
    char name[20];
} *p1, *p2;
```

В якості значень за вказаним адресою призначається адреса об'єкта структури того ж типу:

struct person kate = {31, "Kate"};

struct person *p_kate = &kate;

Використовуючи вказівник на структуру, можна отримати доступ до її елементів. Для цього можна використовувати двома способами.

Перша можливість представляє застосування опису операцій пошуку:

- **(*вказівник _на_структуру).ім'я_елемента**

Друга можливість передбачає використання операцій -> (операція стрілка):

- **вказівник _на_структуру->ім'я_елемента**

Приклад, який поєднує вказівники та структури та служить для введення нового оператора: оператора стрілки (->):

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

```
struct movies_t {
  string title;
  int year;
};
int main ()
{
  string mystr;
  movies_t amovie;
  movies_t * pmovie;
  pmovie = &amovie;
  cout << "Enter title: ";
  getline (cin, pmovie->title);
  cout << "Enter year: ";
  getline (cin, mystr);
  (stringstream) mystr >> pmovie->year;
  cout << "\nYou have entered:\n";
  cout << pmovie->title;
  cout << " (" << pmovie->year << ")\n";

  return 0;}
```

7.3. Порядок виконання роботи

7.3.1. Проаналізувати умову задачі.

7.3.2. Розробити алгоритм та створити програму розв’язання задачі згідно з номером варіанту.

7.3.3. Результати роботи оформити протоколом.

7.4 Варіанти завдань

Варіант 1.

1.Описати структуру з ім'ям STUDENT, що містить наступні поля:

• NAME - прізвище та ініціали;

• GROUP - номер групи;

• SES - успішність (масив з п'яти елементів).

2. Написати програму, що виконує наступні дії:

• введення даних в масив STUD1, що складається з десяти структур типу STUDENT; записи повинні бути впорядковані за зростанням вмісту поля GROUP;

• виведення прізвищ і номерів груп для всіх студентів, включених в масив, якщо середній бал студента більше 4,0;

• якщо таких немає, вивести відповідне повідомлення.

Варіант 2
1. Описати структуру з ім'ям STUDENT, що містить наступні поля:
• NAME - прізвище та ініціали;
• GROUP- номер групи;
• SES- успішність (масив з п'яти елементів).
2. Написати програму, що виконує наступні дії:
• введення даних в масив STUD1, що складається з десяти структур типу STUDENT; записи повинні бути впорядковані за зростанням середнього бала;
• виведення прізвищ і номерів груп для всіх студентів, які мають оцінки 4 і 5;
• якщо таких немає, вивести відповідне повідомлення.

Варіант 3
1. Описати структуру з ім'ям STUDENT, що містить наступні поля:
• NAME - прізвище та ініціали;
• GROUP- номер групи;
• SES- успішність (масив з п'яти елементів).
2. Написати програму, що виконує наступні дії:
• введення даних в масив STUD1, що складається з десяти структур типу STUDENT; записи повинні бути впорядковані за алфавітом;
• виведення прізвищ і номерів груп для всіх студентів, які мають хоча б одну оцінку 2;
• якщо таких студентів немає, вивести відповідне повідомлення.

Варіант 4
1. Описати структуру з ім'ям AEROFLOT, що містить наступні поля:
• NAZN - назва пункту призначення рейсу;
• NUMR - номер рейсу;
• TIP - тип літака.
2. Написати програму, що виконує наступні дії:
• введення даних в масив AIRPORT, що складається з семи елементів типу AEROFLOT; записи повинні бути впорядковані за зростанням номера рейсу;
• виведення на екран номерів рейсів і типів літаків, що відлітають в пункт призначення, назва якого співпало з назвою, введеним ;
• якщо таких рейсів немає, вивести відповідне повідомлення.

Варіант 5

1. Описати структуру з ім'ям AEROFLOT, що містить наступні поля:
• NAZN - назва пункту призначення рейсу;
• NUMR - номер рейсу;
• TIP - тип літака.
2. Написати програму, що виконує наступні дії:
• введення даних в масив AIPORT, що складається з семи елементів типу AEROFLOT; записи повинні бути розміщені в алфавітному порядку за назвами пунктів призначення;
• виведення на екран пунктів призначення і номерів рейсів, що обслуговуються літаком, тип якого введено ;
• якщо таких рейсів немає, вивести відповідне повідомлення.

Варіант 6

1. Описати структуру з ім'ям WORKER, що містить наступні поля:
• NAME - прізвище та ініціали працівника;
• POS - назва займаної посади;
• YEAR - рік надходження на роботу.
2. Написати програму, що виконує наступні дії:
• введення даних в масив TABL, що складається з десяти структур типу WORKER; записи повинні бути розміщені в алфавітному порядку.
• виведення прізвищ працівників, чий стаж роботи в організації перевищує значення, введене ;
• якщо таких працівників немає, вивести відповідне повідомлення.

Варіант 7

1. Описати структуру з ім'ям TRAIN, що містить наступні поля:
• NAZN - назва пункту призначення;
• NUMR - номер поїзда;
• TIME - час відправлення.
2. Написати програму, що виконує наступні дії:
• введення даних в масив RASP, що складається з восьми елементів типу TRAIN; записи повинні бути розміщені в алфавітному порядку за назвами пунктів призначення;
• виведення на екран інформації про поїзди, що відправляються після введеного часу;
• якщо таких поїздів немає, вивести відповідне повідомлення.

Варіант 8
1. Описати структуру з ім'ям TRAIN, що містить наступні поля:
• NAZN - назва пункту призначення;
• NUMR - номер поїзда;
• TIME - час відправлення.
2. Написати програму, що виконує наступні дії;
• введення даних в масив RASP, що складається з шести елементів типу TRAIN; записи повинні бути впорядковані за часом відправлення поїзда;
• виведення на екран інформації про поїзди, що прямують в пункт, назву якого введено;
• якщо таких поїздів немає, вивести відповідне повідомлення.

Варіант 9
1. Описати структуру з ім'ям TRAIN, що містить наступні поля:
• NAZN - назва пункту призначення;
• NUMR - номер поїзда;
• TIME - час відправлення.
2. Написати програму, що виконує наступні дії:
• введення даних в масив RASP, що складається з восьми елементів типу TRAIN; записи повинні бути впорядковані за номерами поїздів;
• виведення на екран інформації про потяг, номер якого введено ;
• якщо таких поїздів немає, вивести відповідне повідомлення.

Варіант 10
1. Описати структуру з ім'ям MARSH, що містить такі, поля:
• BEGST - назва початкового пункту маршруту;
• TERM - назва кінцевого пункту маршруту;
• NUMER - номер маршруту.
2. Написати програму, що виконує наступні дії:
• введення даних в масив TRAFIC, що складається з восьми еле¬ментов типу MARSH; записи повинні бути впорядковані за номерами маршрутів;
• виведення на екран інформації про маршрут, номер якого введено з клавіатури;
• якщо таких маршрутів немає, вивести відповідне повідомлення.

Варіант 11
1. Описати структуру з ім'ям MARSH, що містить наступні поля:
• BEGST - назва початкового пункту маршруту;

• TERM - назва кінцевого пункту маршруту;
• NUMER - номер маршруту.
2. Написати програму, що виконує наступні дії:
• введення даних в масив TRAFIC, що складається з восьми еле¬ментов типу MARSH; записи повинні бути впорядковані за номерами маршрутів;
• виведення на екран інформації про маршрути, які починаються або закінчуються в пункті, назва якого введено ;
• якщо таких маршрутів немає, вивести відповідне повідомлення.

Варіант 12
1. Описати структуру з ім'ям NOTE, що містить наступні поля:
• NAME - прізвище, ім'я;
• TELE - номер телефону;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, що виконує наступні дії:
• введення даних в масив BLOCKNOTE, що складається з восьми елементів типу NOTE; записи повинні бути впорядковані по датах днів народження;
• виведення на екран інформації про людину, номер телефону якого введено ;
• якщо такого немає, вивести відповідне повідомлення.

Варіант 13
1. Описати структуру з ім'ям NOTE, що містить наступні поля:
• NAME - прізвище, ім'я;
• TELE - номер телефону;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, що виконує наступні дії:
• введення даних в масив BLOCKNOTE, що складається з восьми елементів типу NOTE; записи повинні бути розміщені за алфавітом;
• виведення на екран інформації про людей, чиї дні народження припадають на місяць, значення якого введено ;
• якщо таких немає, вивести відповідне повідомлення.

Варіант 14
1. Описати структуру з ім'ям NOTE, що містить наступні поля:
• NAME - прізвище, ім'я;
• TELE - номер телефону;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, що виконує наступні дії:

• введення даних в масив BLOCKNOTE, що складається з восьми елементів типу NOTE; записи повинні бути впорядковані за трьома першими цифрами номера телефону;
• виведення на екран інформації про людину, чиє прізвище введена ;
• якщо такого немає, вивести відповідне повідомлення.

Варіант 15
1. Описати структуру з ім'ям ZNAK, що містить наступні поля:
• NAME - прізвище, ім'я;
• ZODIAC - знак Зодіаку;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, що виконує наступні дії:
• введення даних в масив BOOK, що складається з восьми елементів типу ZNAK; записи повинні бути впорядковані по датах днів народження;
• виведення на екран інформації про людину, чиє прізвище введена з клавіатури;
• якщо такого немає, вивести відповідне повідомлення.

Варіант 16
1. Описати структуру з ім'ям ZNAK, що містить наступні поля:
• NAME - прізвище, ім'я;
• ZODIAC - знак Зодіаку;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, що виконує наступні дії:
• введення даних в масив BOOK, що складається з восьми елементів типу ZNAK; записи повинні бути впорядковані по датах днів народження ;
• виведення на екран інформації про людей, які народилися під знаком, наіменова¬ніе якого введено ;
• якщо таких немає, вивести відповідне повідомлення.

Варіант 17
1. Описати структуру з ім'ям ZNAK, що містить наступні поля:
• NAME - прізвище, ім'я;
• ZODIAC - знак Зодіаку;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, що виконує наступні дії:
• введення даних в масив BOOK, що складається з восьми елементів типу ZNAK; записи повинні бути впорядковані по знакам Зодіаку;

• виведення на екран інформації про людей, які народилися в місяць, значення якого введено ;
• якщо таких немає, вивести відповідне повідомлення.

Варіант 18
1. Описати структуру з ім'ям PRICE, що містить наступні поля:
• TOVAR - назва товару;
• MAG - назва магазину, в якому продається товар;
• STOIM - вартість товару в грн.
2. Написати програму, що виконує наступні дії:
• введення даних в масив SPISOK, що складається з восьми еле¬ментов типу PRICE; записи повинні бути розміщені в алфавітному порядку за назвами товарів;
• виведення на екран інформації про товар, назва якого введено з клавіатури;
• якщо таких товарів немає, вивести відповідне повідомлення.

Варіант 19
1. Описати структуру з ім'ям PRICE, що містить наступні поля:
• TOVAR - назва товару;
• MAG - назва магазину, в якому продається товар;
• STOIM - вартість товару в грн.
2. Написати програму, що виконує наступні дії:
• введення даних в масив SPISOK, що складається з восьми еле¬ментов типу PRICE; записи повинні бути розміщені в алфавітному порядку за назвами магазинів;
• виведення на екран інформації про товари, що продаються в магазині, назва якого введено ;
• якщо такого магазину немає, вивести відповідне повідомлення.

Варіант 20
1. Описати структуру з ім’ям ORDER, що містить наступні поля:
• PLAT - розрахунковий рахунок платника;
• POL - розрахунковий рахунок одержувача;
• SUMMA - перераховується сума в грн.
2. Написати програму, що виконує наступні дії:
• введення даних в масив SPISOK, що складається з восьми еле¬ментов типу ORDER; записи повинні бути розміщені в алфавітному порядку по розрахункових рахунках платників;

• виведення на екран інформації про суму, зняту з розрахункового рахунку платника, введеного ;
• якщо такого розрахункового рахунку немає, вивести відповідне повідомлення.

7.5 Контрольні запитання

1. Опишіть порядок створення структури
2. Як виділяється пам’ять під структуру
3. Як отримати доступ до елементів структури
4. Наведіть приклад передачі структури в функцію
5. Як створити масив структур тв отримати доступ до останнього елементу масиву
6. Поясніть на прикладі використання . та ->

Лабораторна робота №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

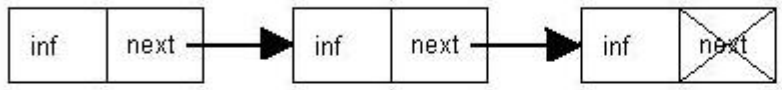
Зв'язний список — це структура даних або вузлів, кожний з яких містить як власні дані, так і один або два вказівника («посилання») на наступний та/або попередній вузол. Принциповою перевагою таких конструкцій перед звичайним масивом є їх гнучкість, оскільки порядок елементів зв'язного списку може не 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.

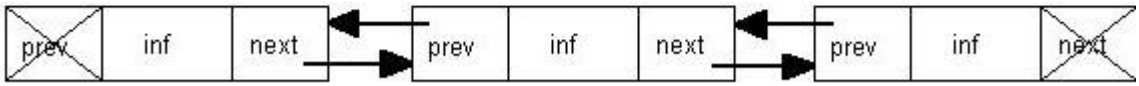
Кожний елемент зв'язного списку є окремим об'єктом (або структурою), що містить поля для зберігання інформації та вказівник на наступний елемент списку. (Якщо список є двозв'язним, то в такому об'єкті зберігається також вказівник на попередній елемент.)

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

Связный список



Двусвязный список

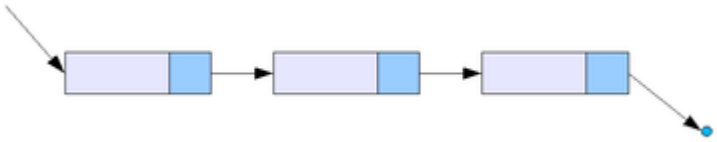


Пересування по списку здійснюється за вказівниками, які містять адресу сусіднього елемента списку. При додаванні до списку нового елемента необхідно динамічно виділити для нього пам'ять за допомогою оператора new та присвоїти відповідні адреси вказівникам сусідніх елементів.

Види зв'язних списків

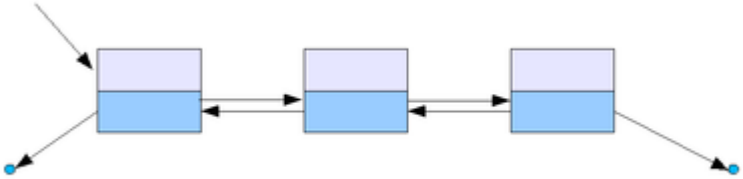
Існує декілька можливих типів зв'язних списків. Найбільш поширені з них розглядаються у цьому розділі.

Однозв'язний список



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

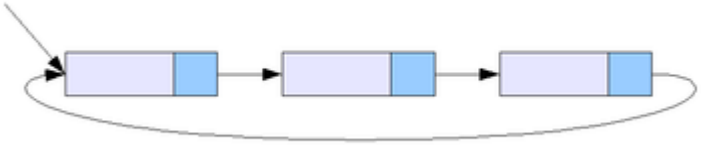
Двозв'язний список



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 елеменів списку.

Кільцевий зв'язний список

Різновидом зв'язних списків є кільцевий (або циклічний, замкнутий) 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.



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 функціональість надає утиліта make.

В той же час 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 обидвох 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, легко випустити з уваги, що у випадку змін необхідно вносити відновлення також в обидва додаткових наборів оголошень.

Ця проблема легко розв’язується за допомогою додаткового заголовного файла. 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 раніш реалізовані функції, достатньо скористатися існуючим заголовним файлом і додати до складу відповідного проекту файл із реалізацією ціх 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 слід додавати заголовні файли до *списку проекту* (project list). Краще також не використовувати директиву #include для включення одних файлів програмного коду в інші файли.

Більш докладно теоретичні питання, пов'язані з даною лабораторною работою, викладено в [Прата], [Шилдт], [Гаврилова-Рецензент], [Айри Підлога].

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 принципа роздільної компіляції. До складу проекту проекту входить три файли: перший містить реалізацію головної функції, два інших складають пару «заголовний файл—файл з реалізацією», що забезпечує можливість легкого підключення бібліотеки функцій у будь-яку програму на мові C++.

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

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;

     getch();
     return 0;
}
```

Для створення виконуваної програми потрібно додати до її проекту два файла 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 файла (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.

Вимоги до вмісту протоколу

1. 1 сторінка — титульний лист стандартного вигляду (вказати № залікової книжки);
2. 2 сторінка — умова задачі, яка розв’язується;
3. блок-схема алгоритму розв’язання задачі;
4. таблиця з переліком змінних (разом з їх типом), типів даних користувача та їх призначенням;
5. опис проектних рішень, які, за думкою виконавця, необхідно описати більш детально.
6. програмний код;

8.4. Варіанти завдань

Виконати **будь-які 2 завдання** (на вибір) із переліку базових, або підвищеної складності.

Базові завдання

1. Створити функцію, яка обчислює довжину списку для будь-якого типу його елементів (для шаблонного варіанту типу вузлів списку).

2. Створити функцію, яка повертає вказівник на другий елемент списку.

3. Створити функцію, яка повертає вказівник на останній елемент списку.

4. Створити функцію, яка повертає вказівник на елемент, що знаходиться в заданій позиції списку (додається один цілий параметр - номер необхідної позиції; 0 - перший елемент списку, 1 - другий елемен, і т.д.,): а) для списку цілих чисел; б) для списку з будь-якого типу елементів.

5. Створити функцію, що виводить всі елементи списку на екран.

6. Створити функцію, яка підраховує кількість елементів списку, що задовольняють певній умові (додається один параметр - вказівник на функцію, що приймає елемент даних і повертає логічне значення; розглядати потрібно ті елементи, на яких ця функція повертає true).

7. Створити функцію, яка повертає вказівник на: а) перший; б) останній елемент списку з зазначеними даними (другий параметр функції).

Завдання підвищеної складності

1. Написати функцію додавання першого елементу в список для будь-якого типу його елементів (для шаблонного варіанту типу вузлів списку).

2. Написати функцію додавання останнього елементу в список для будь-якого типу його елементів (для шаблонного варіанти типу вузлів списку).

3. Написати функцію додавання елемента в задану позицію списку (додається один цілий параметр - номер необхідної позиції; 0 - вставка в початок списку, 1 - після першого елемента, і т.д.): а) для списку цілих чисел; б) для списку з будь-якого типу елементів.

4. Написати функцію без параметрів, що зчитує елементи списку з клавіатури до тих пір, поки елементи не закінчаться (кінець потоку) і повертає вказівник на перший елемент створеного списку: а) за допомогою операції вставки елемента в кінець списку (близько n^2 дій, де n - довжина списку); б) більш швидкий варіант – близько n дій (використовуючи допоміжний покажчик на поточний вузол).

5. Написати функцію, яка будує копію списку (при побудові вузли створюються за допомогою new), що містить ту ж саму послідовність даних в вузлах) рекурсивно з використанням конструктора вузла.

6. Створити копію списку, але розмістити елементи в зворотному порядку.

8.5. Контрольні запитання

1. Які переваги та недолікимають 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. http://cpp.dp.ua/ «Основи розробки алгоритмів та програмування мовою C++». Електронний посібник
2. http://www.znannya.org/?view=Cpp_basics - Огляд і основи мови програмування C++.
3. https://stepik.org/lesson/534/step/3?unit=857 – Структура коду на C++
4. https://stepik.org/lesson/538/step/2?unit=861 – Стек викликів
5. https://www.youtube.com/watch?v=ohW8iQQaZYY – Робота з файлами
6. https://www.stroustrup.com/Programming/PPP2_Ch18.pdf - STL, Робота з векторами та масивами
7. https://www.programiz.com/cpp-programming