Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Кафедра програмних та комп'ютерно-інтегрованих технологій

# Методичні вказівки з дисципліни «Програмування та теорія алгоритмів».

(Теоретична частина)

Для студентів інституту штучного інтелекту та робототехніки

Перший (бакалаврський) рівень вищої освіти

Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології

Освітньо-професійна програма: Комп'ютерні технології автоматизації.

Одеса 2022

Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Кафедра програмних та комп'ютерно-інтегрованих технологій

Методичні вказівки з дисципліни «Програмування та теорія алгоритмів».
(Теоретична частина)
Для студентів інституту штучного інтелекту та робототехніки

Перший (бакалаврський) рівень вищої освіти
Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо-професійна програма: Комп'ютерні технології автоматизації.

Затверджено на засіданні
кафедри програмних та комп'ютерно-інтегрованих технологій
Протокол № 7 від 26.01.2022р.

Одеса 2022

Методичні вказівки з дисципліни Програмування та теорія алгоритмів. (Теретична частина): для студ. напряму 151 «Автоматизація та комп'ютерно-інтегровані технології» денної та заочної форм навчань./ Укл. Лисюк Г.П. - Одеса: ОП, 2022. - 70 с.

**ЗМІСТ**

1. Робота з файлами. ....4
2. Робота з рядками. ....10
3. Робота з бінарними файлами. ....22
4. Структури. ....27
5. Класи. ....32
6.Управління доступом. ....36
7. Конструктор класу. ....41
8. Деструктор класу. Ключове слово *this.* ....45
9. Використання списку ініціалізації у конструкторі. Дружні функції класу. ....50
10. Спадкування класів та віртуальні функції. ....53
11. Віртуальна функція. Поліформізм. ....64

## 1. РОБОТА З ФАЙЛАМИ.

Більшість комп'ютерних програм працюють із файлами, і тому виникає потреба створювати, видаляти, записувати читати, відкривати файли.

Файл - іменований набір байтів, який може бути збережений на деякому накопичувачі, яка має своє унікальне ім'я, наприклад файл.txt. В одній директорії не можуть бути файли з однаковими іменами. Під ім'ям файлу розуміється не тільки його назва, а й розширення, наприклад: file.txt і file.dat-різні файли, хоч і мають однакові назви. Існує таке поняття, як повне ім'я файлів – це повна адреса до директорії файлу із зазначенням імені файлу, наприклад: D:\docs\file.txt.

Для роботи з файлами необхідно засвоїти таке поняття як потік – абстрактний канал зв'язку, що створюється у програмі обмінюватись даними з файлами.

*Текстовий потік*- Це послідовність символів. Під час передачі символів із потоку на екран частина з них не виводиться (наприклад, символ повернення каретки, перекладу рядка).

*Двійковий потік*- Це послідовність байтів, які однозначно відповідають тому, що знаходиться на зовнішньому пристрої.

Розглянемо роботу із текстовими файлами.

C++ відсутні оператори для роботи з файлами. Усі необхідні дії виконуються за допомогою функцій, що включені до стандартної бібліотеки. Вони дозволяють працювати з різними пристроями, такими як диски, принтер, комунікаційні канали і т.д. Ці пристрої дуже відрізняються один від одного. Однак файлова система перетворює їх на єдиний абстрактний логічний пристрій, званий потоком.

Для роботи з файлами необхідно підключити заголовний файл <fstream>**.** В <fstream> визначено кілька класів та підключено заголовні файли <ifstream> -файлове введення та <ofstream> -файловий висновок.

Файлове введення/виведення аналогічне стандартному введення/виводу, єдина відмінність – це те, що введення/виведення виконуються не на екран, а у файл. Якщо введення/виведення на стандартні пристрої виконується за допомогою об'єктів cin і cout, то для організації файлового введення/виводу достатньо створити власні об'єкти, які можна використовувати аналогічно операторам cin і cout.

Наприклад, необхідно створити текстовий файл та записати в нього рядок Робота з файлами в C++. Для цього необхідно зробити наступні кроки:

1. створити об'єкт класу ofstream;
2. зв'язати об'єкт класу з файлом, в який буде записуватися;
3. записати рядок у файл;
4. закрити файл.

Чому необхідно створювати об'єкт класу ofstream, а не класу ifstream? Тому що потрібно зробити запис у файл, а якби потрібно було рахувати дані з файлу, то створювався б об'єкт класу ifstream.

```
// створюємо об'єкт для запису у файл
ofstream /*ім'я об'єкта*/; // об'єкт класу ofstream
```

Назвемо об'єкт – fout, Ось що вийде:

```
ofstream fout;
```

Навіщо нам об'єкт? Об'єкт необхідний, щоб виконувати запис у файл. Вже об'єкт створено, але не пов'язаний із файлом, у який потрібно записати рядок.

```
fout.open("cppstudio.txt"); // зв'язуємо об'єкт із файлом
```

Через операцію точка отримуємо доступ до методу класу open(), у круглих дужках якого вказуємо ім'я файлу. Зазначений файл буде створено у поточній директорії з програмою. Якщо файл з таким ім'ям існує, існуючий файл буде замінено на новий. Отже, файл відкритий, залишилося записати до нього потрібний рядок. Робиться це так:

```
fout << "Робота з файлами в С++"; //запис рядка у файл
```

Використовуючи операцію передачі в потік спільно з об'єктом fout рядок Робота з файлами в С++ записується у файл. Оскільки більше не потрібно змінювати вміст файлу, його потрібно закрити, тобто відокремити об'єкт від файлу.

```
out.close(); // закриваємо файл
```

Підсумок – створено файл із рядком Робота з файлами в С++.

Кроки 1 та 2 можна об'єднати, тобто в одному рядку створити об'єкт та зв'язати його з файлом. Робиться це так:

```
ofstream fout("cppstudio.txt"); // створюємо об'єкт класу ofstream
та зв'язуємо його з файлом cppstudio.txt
```

Об'єднаємо весь код та отримаємо наступну програму.

```
#include <fstream>
using namespacestd;

intmain()
{
 ofstreamfout("cppstudio.txt");// створюємо об'єкт класу
 // ofstream для запису та пов'язуємо його з файлом cppstudio.txt
 /*
 ofstream fout;
fout.open("cppstudio.txt");
*/

fout<< "Робота з файлами в С++";// запис рядка у файл
fout.close();// закриваємо файл
 return0;
}
```

Залишилося перевірити правильність роботи програми, а для цього відкриваємо файл cppstudio.txti дивимося його вміст, мабуть Робота з файлами в С++.

Для того, щоб прочитати файл, потрібно виконати ті ж кроки, що і при записі в файл з невеликими змінами:

1. створити об'єкт класу ifstream і зв'язати його з файлом, з якого проводитиметься зчитування;
2. прочитати файл;
3. закрити файл.

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespacestd;

intmain()
{
setlocale(LC_ALL,"rus");// Коректне відображення Кирилиці
```

```
 charbuff[50];// буфер проміжного зберігання зчитуваного // файлу тексту
 ifstreamfin("cppstudio.txt");// відкрили файл для читання
fin>>buff;// Вважали перше слово з файлу
cout<<buff<<endl;// надрукували це слово
fin.getline(buff, 50);// Вважали рядок з файлу
fin.close();// закриваємо файл
cout<<buff<<endl;// надрукували цей рядок

 return0;
}
```

У програмі показано два способи читання з файлу, перший – використовуючи операцію передачі в потік, другий – використовуючи функцію getline(). У першому випадку зчитується лише перше слово, а у другому випадку зчитується рядок, довжиною 50 символів. Але оскільки у файлі залишилося менше 50 символів, то символи зчитуються включно до останнього.

**Загальна схема організації введення-виводу така:**

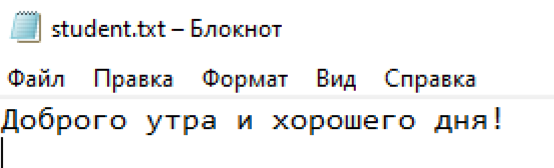
1. Створити об'єкт організації вводу-вывода.

2. Переконатися, що об'єкт створено успішно (за допомогою методу is_open()).

3. Виконати необхідні операції з введення даних (за допомогою оператора >>) або щодо виведення даних (за допомогою оператора <<).

4. Закінчити роботу з об'єктом (за допомогою методу close()).

**Приклад_1.1:**
//Вивести на екран вміст файлу student.txt




**Рішення:**

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespacestd;

intmain()
{
 ifstreamFin("student.txt");
 if(Fin.is_open())
{
        setlocale(LC_ALL,"rus");//коректне відображення Кирилиці
         charStr[50];
        Fin>>Str;
        cout<<Str;
        Fin.getline(Str, 50); метод файлу
```

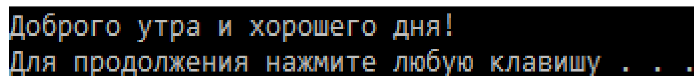
```
		cout<<Str<<'\n';
		Fin.close();
	}
	 else
	{
		cerr<< "Помилка введення файлу";
	}
	}
```

**cerr**- стандартний потік помилок без буферизації, який використовується для виведення помилок.

Це також екземпляр класу ostream.

Оскільки cerr небуферизований, він використовується, коли нам потрібно негайно відобразити

повідомлення про помилку.




**Приклад_1.2**

```
//Запис у текстовий файл із 10 цифр.
#include <iostream>
#include <fstream>
#include <cstring>
#include <time.h>
using namespacestd;
intmain()
{
 ofstreamFout;
Fout.open("array.txt");
 if(Fout.is_open())
{
	srand(time(NULL));
	setlocale(LC_ALL,"rus");//коректне відображення Кирилиці
	inti;
	 intArray[10];
	 for(i = 0; i < 10; i++)
	{
		Array[i]= rand() % (10 + 1) - 5;
		Fout<<Array[i]<<"\n";
		cout<< '\t' <<Array[i];
	}
	cout<<endl;
Fout.close();
}
 else
{
cerr<< "Помилка введення файлу";
}
```

```
}
```

**Приклад_1.3**

```
//Читання даних із текстового файлу в масив.

#include <iostream>
#include <fstream>
#include <cstring>
using namespacestd;

intmain()
{
 ifstreamFin;
Fin.open("array.txt");
 if(Fin.is_open())
{
setlocale(LC_ALL,"rus");//коректне відображення Кирилиці
 inti;
 intArray[10];
 for(i = 0; i < 10; i++)
{
Fin>>Array[i];
cout<< '\t' <<Array[i];
}
cout<<endl;
Fin.close();
}
 else
{
cerr<< "Помилка введення файлу";
}
}
```

**Приклад_1.4.**

//Дано функцію: y = x2. Провести функцію табуляції на інтервалі [a,b] з кроком Δx = (b - a)/10. Занести результат до текстового файлу.

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <time.h>
#include <math.h>
using namespacestd;

intmain()
{
 ofstreamFout;
                    // Відкриваємо потік
Fout.open("array_2.txt");
 If (Fout.is_open())
{
setlocale(LC_ALL,"rus");// Коректне відображення Кирилиці
 inti,a,b;
```

```
cout<< "a="; cin>>a; cout<<endl;
cout<< "b="; cin>>b; cout<<endl;
// Перевірка коректності введених даних
        if ( b < = a )
        {
                cout << "Wrong inteval\n";
                return;
        }

 doubledx, x, y;
                // Розрахунок кроку аргументу
dx = (b – a)/10.;
 for(i = 0; i <= 10; i++)
{
x = a + i * dx;
y=pow(x,2);
                        // Запис у файл чергового значення аргументу та функції
Fout<<x<< "\t"<<y<<'\n';
// Виведення на екран значення аргументу та функції
cout<<x<< "\t" <<y<< '\n';
}
Fout.close();
}
 else
{
cerr<< "Помилка введення файлу";
}
}
```

```
a= 2

b= 6

2       4
2.4     5.76
2.8     7.84
3.2     10.24
3.6     12.96
4       16
4.4     19.36
4.8     23.04
5.2     27.04
5.6     31.36
6       36
```

**Приклад_1.5.**

//Прочитати з файлу значення табульованої функції, отримані у попередній задачі та вивести їх на екран.

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespacestd;
```

```
intmain()
{
 ifstreamFin;
Fin.open("array_2.txt");
 if(Fin.is_open())
{
setlocale(LC_ALL,"rus");//коректне відображення Кирилиці
 inti;
                              // Масив для зберігання значень аргументу та функції
 doubleArray[11][2];
 for(i = 0; i <= 10; i++)
{
Fin>>Array[i][0];//Читаємо в масив значення аргументу
Fin>>Array[i][1];// Читаємо в масив значення функції
 // Виводимо дані на екран
cout<<Array[i][0]<< '\t'<<Array[i][1]<<'\n';
}
 // Закриваємо потік
Fin.close();
}
 else
{
cerr<< "Помилка введення файлу";
}
```

## 2. РОБОТА З РЯДКАМИ.

Символьна константа – це цілісне значення (типуint) представлене у вигляді символу, укладеного в одинарні лапки, наприклад'a'. У таблиці ASCII представлені символи та їх цілі численні значення.

```
// оголошення символьної змінної
char symbol = 'a'; // symbol – ім'я змінної типу char
```

**C++ підтримує два різні типи рядків:**
std::string (як частина Стандартної бібліотеки C++);
рядки C-style (спочатку успадковані від мови Сі).
std::string реалізований за допомогою рядків C-style.

**Рядки C-style**
Щоб користуватися бібліотечними функціями для роботи з рядками, потрібно підключити файл cstring.

```
#include <cstring>
```

Рядки C++ представляються як масиви елементів типуchar, що закінчуються нуль-термінатором\0називаються З рядками або рядками у стилі C.

\0- Символ нуль-термінатора.

При оголошенні рядкового масиву необхідно враховувати наявність кінці кінці нуль-термінатора, і відводити додатковий байт під нього.

Рядок може містити символи, цифри та спеціальні знаки. У C++ рядки полягають у подвійні лапки. Ім'я рядка є константним вказівником на перший символ.

Оголосити рядкову змінну в такому стилі можна двома способами:

а) як покажчик:

char * s;

s = new char[число_символів_рядка+1];

Тут +1 потрібен для зберігання нульового символу, що завершує рядок. При використанні динамічних масивів потрібно дбати про своєчасне їхнє звільнення операцією delete. Однак таку змінну можна ініціалізувати рядковою константою:

char *s = "текст_рядка";

Така можливість існує завдяки тому, що C++ рядкова константа — це просто вказівник на перший символ рядка, укладеного в лапки (завершальний нульовий символ додається автоматично компілятором). Однак, звичайно рядкові константи зберігаються в тій частині пам'яті, куди не можна нічого записувати, так що у так певного рядка не можна змінювати символи, що входять до її складу. Якщо ж ми хочемо, щоб вміст рядка міг змінюватися, потрібно користуватися масивами (динамічні або звичайні).

б) як масив:

char s [максимальне_число_символів +1];

```
// приклад оголошення рядка
char string[10]; // string - ім'я рядкової змінної, 10 - розмір масиву,
// тобто в даному рядку може поміститися 9 символів,
// Останнє місце відводиться під нуль-термінатор.
```

Рядок при оголошенні може бути ініціалізований початковим значенням, наприклад, так:

```
char string[10] = "abcdefghf";
```

Якщо підрахувати кількість символів у подвійних лапках після символу і їх виявиться 9, а розмір рядка 10 символів, останнє місце відводиться під нуль–термінатор, причому компілятор сам додасть їх у кінець рядка.

```
// Посимвольна ініціалізація рядка:
char string[10] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'f', '\0'}; // десятий символ це
// Нуль-термінатор.
```

При оголошенні рядка не обов'язково вказувати її розмір, але заодно обов'язково потрібно її ініціалізувати початковим значенням. Тоді розмір рядка визначиться автоматично і до кінця рядка додасться нуль-термінатор.

```
//ініціалізація рядка без зазначення розміру
char /tring[] = "abcdefghf"; //все те саме розмір не вказуємо.
```

Щоб користуватися бібліотечними функціями для роботи з рядками, потрібно підключити файл cstring.

```
#include <cstring>
```

***Доступ до окремого символу.***

Як і для будь-якого масиву, доступ до символу з рядка s за індексом (починається з 0) виглядає як

Str [індекс].

**Приклад_2.1.**

//Ініціалізація рядка.

//Визначення довжини рядка, включаючи нуль-термінатор (sizeof(Str)).

/Визначення довжини рядка без нуль-термінатора (strlen(Str)).
//Виведення термінів символів та їх кодів.

```
#include <iostream>
#include <cstring>
voidmain()
{
charStr[] ="Good day!";//ініціалізація рядка без зазначення розміру
std::cout<<Str<< " length="<<sizeof(Str)<<" characters" << '\n';
//Довжина рядка:
std::cout<<Str<< " length=" <<strlen(Str)<< " characters" << '\n';
 inti;
  for(i = 0; i <sizeof(Str); i++)
{
std::cout<<Str[i]<< '\t';
}
std::cout<<std::endl;

 for(i = 0; i <sizeof(Str); i++)
{
std::cout<< static_cast<int>(Str[i])<< '\t';

}
std::cout<<std::endl;
//Доступ до окремого символу.
std::cout<< "Str[3] = "<<Str[3]<<std::endl;
}
```

```
Good day! length=10 characters
Good day! length=9 characters
G       o       o       d               d       a       y       !
71      111     111     100     32      100     97      121     33      0
Str[3] = d
```

**Приклад_2.2.**

//Програма виводить 128 символів ASCII таблиці (десятизначне число та його символ). Так щоб у ній було 16 рядків та 8 стовпців, і символи виводилися по стовпчиках зверху донизу.

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
char s[16][8];
int i, j, x=0, mas[16][8];
for(i=0; i<8; i++)
{
   for (j=0; j<16; j++)
   {
               mas[j][i]=x;
               s[j][i]=x;
```

```
                x++;
        }
    }
    for(i=0; i<16; i++)
    {
      for (j=0; j<8; j++)
        {
      //Умова для більше рівного висновку таблиці
                if (mas[i][j]==9)
                        cout<<mas[i][j]<<" = "<<s[i][j];
                else
                        cout<<mas[i][j]<<" = "<<s[i][j]<<"\t";
        }
      cout<<endl;
    }
    return 0;
    }
```

```
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
0 =      16 = ►  32 =    48 = 0  64 = @  80 = P  96 = `  112 = p
1 = ☺    17 = ◄  33 = !  49 = 1  65 = A  81 = Q  97 = a  113 = q
2 = ☻    18 = ↕  34 = "  50 = 2  66 = B  82 = R  98 = b  114 = r
3 = ♥    19 = ‼  35 = #  51 = 3  67 = C  83 = S  99 = c  115 = s
4 = ♦    20 = ¶  36 = $  52 = 4  68 = D  84 = T  100 = d 116 = t
5 = ♣    21 = §  37 = %  53 = 5  69 = E  85 = U  101 = e 117 = u
6 = ♠    22 = ▬  38 = &  54 = 6  70 = F  86 = V  102 = f 118 = v
7 =      23 = ↨  39 = '  55 = 7  71 = G  87 = W  103 = g 119 = w
8 =      24 = ↑  40 = (  56 = 8  72 = H  88 = X  104 = h 120 = x
9 =      25 = ↓  41 = )  57 = 9  73 = I  89 = Y  105 = i 121 = y
10 =
         26 = →  42 = *  58 = :  74 = J  90 = Z  106 = j 122 = z
11 = ♂   27 = ←  43 = +  59 = ;  75 = K  91 = [  107 = k 123 = {
12 = ♀   28 = ∟  44 = ,  60 = <  76 = L  92 = \  108 = l 124 = |
         29 = ↔  45 = -  61 = =  77 = M  93 = ]  109 = m 125 = }
14 = ♫   30 = ▲  46 = .  62 = >  78 = N  94 = ^  110 = n 126 = ~
15 = ☼   31 = ▼  47 = /  63 = ?  79 = O  95 = _  111 = o 127 = ⌂
```

***Операції над рядками.***

Рядки в стилі С не можна присвоювати, ні порівнювати за допомогою звичайних операцій порівняння.

Приклади деяких функцій:.

| Функція | Пояснення |
|---|---|
| strlen(ім'я_рядки) | визначає довжину зазначеного рядка, без урахування нуль-символу |
| **Копіювання рядків** | |
| **strcpy_s**(s1,s2) | виконує побайтове копіювання символів з рядка s2 до рядка s1 |
| **strncpy_s**(s1, s2, n) | виконує побайтове копіювання n символів з рядка s2 до рядка s1. повертає значення s1 |
| **Конкатенація рядків** | |

| | |
|---|---|
| **strcat_s**(s1,s2) | поєднує рядок s2 із рядком s1. Результат зберігається в s1 |
| **strncat_s**(s1, s2, n) | поєднує n символів рядка s2 із рядком s1. Результат зберігається в s1 |

**Приклад_2.3.**

```
#include <iostream>
#include <cstring>
 using namespacestd;
 intmain()
{
 charstring[] ="i am happy!";

 charinstring[50];
 charinstring1[50];

 intc;
c = strlen (string);//Довжина рядка
cout<<c<<endl;
cout<< "*****" <<endl;

strcpy_s(instring, string);//копіювання в 1 рядок
cout<<string<<endl<<instring<<endl;
cout<< "*****" <<endl;

strncpy_s(instring1, instring, 7);//Копір в 1 рядок 7 симв
cout<<instring<<endl<<instring1<<endl;
cout<< "*****" <<endl;

strcat_s(instring, string);//об'єднання рядків у 1 рядку
cout<<instring<<endl;
cout<< "*****" <<endl;

strncat_s(instring, string, 3);//про 1 рядок з сим 2 стор
cout<<instring<<endl;

 for(inti = c – 1; i >= 0; i--)
{
cout<<string[i];
}
 return0;
}
```

```
11
*****
i am happy!
i am happy!
*****
i am happy!
i am ha
*****
i am happy!i am happy!
*****
i am happy!i am happy!i a
!yppah ma iДля продолжения нажмите любую клавишу . . .
```

**Метод getline.**

Вважати рядок до символу перекладу рядка (у тому числі з пробілами) можна за допомогою методу getline:

```
cin.getline(s, розмір_масиву)
using namespace std;

int main()
{
char str [256];

cout << "Введіть чотири пропозиції: " << endl;
cin.getline(str, 256, ';');
cout << str;
cin.getline(str, 256, ';');
cout << str;
}
```

**Тип STRING.**

Тип string дозволяє нам працювати з рядками. Щоб мати можливість використовувати рядки C++, спочатку потрібно підключитизаголовний файл string. Як тільки це буде зроблено, ми зможемо визначати змінні типи string:

```
#include <string>

// ...
std::string name;
// ...
```

Відразу зазначимо, що заголовний файл string входить різні інші бібліотеки, наприклад, iostream.

Як і зі звичайними змінними, ми можемо ініціалізувати змінні типи string або надавати їм значення:

```
std::string name ("Sasha");// ініціалізуємо змінну name рядковим літералом "Sasha"
name ="Masha";// привласнюємо змінною name рядковий літерал "Masha"
```

Рядки можна виводити за допомогою **std::cout**:

```
#include <iostream>
```

```
intmain()
{
	std::string name("Sasha");
	std::cout<< "My name is " <<name;
	return0;
}
```

**Приклад_2.4.**
//Робота з рядками:

```
#include <iostream>
#include <string>
using namespacestd;
intmain()
{
 stringstr1 ="abcdefgh";
 stringstr2 ="kkkkkkkkk";
 stringstr3;
str3=str1;
cout<< "str3=" <<str3<<endl;

 // конкатенація рядків
str3=str3+str2;
cout<< "str3=" <<str3<<endl;

 //доступ до окремого символу
cout<<str1.at(1)<<endl;

 // З'ясовуємо, чи не порожній рядок.
 if(str1.empty())
{
cout<< "String is empty" << "\n";
}
 else{
cout<< "String isn't empty" << "\n";
}

 // Обмін значення двох рядків.
swap(str1, str2);
cout<< "str1=" <<str1<<endl;
cout<< "str2=" <<str2<<endl;

 // Привласнюємо та порівнюємо 2 рядки.
str3=str1;
 if(str3==str1)
{
cout<< "Strings are equal" <<endl;
}
 else{
cout<< "Strings are not equal" <<endl;
}
```

```
// Читання введеного з клавіатури рядка.
cout<< "input string" <<endl;
cin>>str3;
cout<< "str3=" <<str3<<endl;
cout<< "str1=" <<str1<<endl;
cout<< "str2=" <<str2<<endl;

// Отримання довжини рядка.
cout<<str3.length()<<endl;
 return0;
}
```

```
str3 = abcdefgh
str3 = abcdefghkkkkkkkkk
b
String isn't  empty
str1 = kkkkkkkkk
str2 = abcdefgh
Strings are equal
input string
ZZZZZZZZZZZZZ
str3 = ZZZZZZZZZZZZZ
str1 = kkkkkkkkk
str2 = abcdefgh
13
Для продолжения нажмите любую клавишу . . .
```

Мінімальний набір операцій, якими повинен мати клас string:

- ініціалізація масивом символів.
- копіювання одного рядка в інший. .
- доступ до окремих символів рядка для читання та запису. .
- порівняння двох рядків на рівність. .
- конкатенація двох рядків.
- обчислення довжини рядка. .
- можливість дізнатися, чи порожній рядок. .

Клас string стандартної бібліотеки C++ реалізує всі ці операції (і набагато більше).

Для того щоб використовувати об'єкти класу string, необхідно увімкнути відповідний заголовний файл:

```
#include <string>
```

```
string st("a am very clever\n");
```

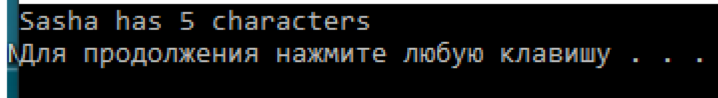
***Визначення довжини рядка:***

**Довжину рядка**повертає функцію-член size() (довжина не включає завершальний нульовий символ).

```
cout << "Length "
          << st
          << ": " << st.size()
```

```
<< " characters, including newline \n";
```

```
#include <iostream>
intmain()

{
        std::stringmyName("Sasha");
        std::cout<<myName<< "has" <<myName.length()<< " characters\n";
        return0;
}
```



Sasha has 5 characters
Для продолжения нажмите любую клавишу . . .

***Визначення, чи порожній рядок:***

Друга форма визначення рядка задає порожній рядок:

**string st2;**// порожня стрічка

Як ми дізнаємося, чи порожній рядок?

Спеціальний метод empty(), який повертає true для порожнього рядка і false для непустого:

```
if ( st.empty() )
// Правильно: порожня
```

Ініціалізація об'єкта типу string іншим об'єктом того ж типу:

**string st3(st);**

Рядок st3 ініціалізується рядком st. Як ми можемо переконатись, що ці рядки збігаються? Скористаємося оператором порівняння (==):

```
if ( st == st3 )
// ініціалізація спрацювала
```

***Копіювання одного рядка в інший:***

За допомогою звичайної операції присвоєння:

st2 = st3; // Копіюємо st3 в st2

***Конкатенація рядків:***

**Для конкатенації рядків**використовується операція додавання (+) або операція додавання з присвоєнням (+=). Нехай дані два рядки:

```
string s1("hello,");
string s2("world\n");
```

Ми можемо отримати третій рядок, який складається з конкатенації перших двох, таким чином:

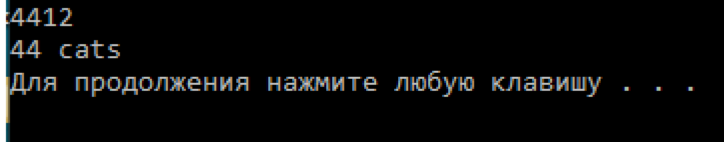
string s3 = s1 + s2;

Якщо ж ми хочемо додати s2 до кінця s1, ми повинні написати:

s1 + = s2;

```
#include <iostream>
intmain()
{
```

```
        std::stringx("44");
        std::stringy("12");

        std::cout<<x+y<< "\n";// об'єднуємо рядки x та y (а не додаємо числа)
        x+= "cats";
        std::cout<<x;

        return0;
}
```




До окремих символів об'єкта типу string, як і вбудованого типу можна звертатися за допомогою операції отримання індексу. Ось, наприклад, фрагмент коду, який замінює всі крапки символами підкреслення:

```
string str("fa.disney.com");
  int size = str.size();

  for ( int ix = 0; ix < size; ++ix )
          if ( str [ ix ] == '.' )
                  str [ix] = '_';
```

**Функція getline із типом string**

**Використання std::getline()**

Щоб отримати повний рядок із вхідного потоку даних (разом із пробілами), використовуйте функцію std::getline().

Вона приймає два параметри: перший – std:: cin, другий – змінна типу string.

**Приклад_2.5:**

```
#include <iostream>
#include <string>

int main()
{
std::cout << "Enter your full name: ";
std::string myName;
std::getline(std::cin, myName); // повністю вилучаємо рядок змінну myName

std::cout << "Enter your age: ";
std::string myAge;
std::getline(std::cin, myAge); // повністю вилучаємо рядок у змінну myAge

std::cout << "Your name is " << myName << " and your age is " << myAge;
}
```

**Enter your full name: Sasha Mak**
**Enter your age: 25**
**Your name is Sasha Mak and your age is 25**

**getline(stream, string, separator);**

де stream – це потік даних,

string - змінна, в яку запишеться рядок і

separator – рядковий роздільник, який показує кінець рядка. Останній параметр функції можна опустити, тоді буде заданий стандартний сепаратор - '\n'.

**Приклад_2.6.**

```
#include <iostream>
#include <string>

using namespacestd;

string check_pass (string password)
{
 stringvalid_pass ="qwerty123";
 stringerror_message;
 if(password ==valid_pass)
{
error_message= "Доступ дозволений.";
}
 Else
{
error_message= "Невірний пароль!";
}
 return error_message;
}

int main()
{
setlocale(LC_ALL,"Український");
 string user_pass;
cout<< "Введіть пароль: ";
getline(cin, user_pass);
 string error_msg = check_pass(user_pass);
cout<<error_msg<<endl;
 return 0;
}
```

```
Введите пароль: qwerty123
Доступ разрешен.
Для продолжения нажмите любую клавишу . . .
```

```
Введите пароль: 11111
Неверный пароль!
Для продолжения нажмите любую клавишу . . .
```

**Приклад_2.7.**

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

```
void main()
{
setlocale(LC_ALL,"Український");
 ofstream Fout;
Fout.open("FileStr.txt");
 If (Fout.is_open())
{
cout<< "Файл відкритий для запису" <<endl;
 string Str;
 charans ='1';
 while(ans =='1')
{
	cout<< "Введіть слово:";
	cin>>Str;
	Fout<<Str<< '\n';
	cout<<endl;
	cout<< " Будете продовжувати? Натисніть 1 , услі так ";
	cin>>ans;
	cout<<endl;

}
Fout.close();
}
 else
{
	cerr<< "Помилка введення файлу";
}

 Ifstream Fin;
Fin.open("FileStr.txt");
 String Str;
 If (Fin.is_open())
{
	cout<< "Файл відкритий для читання"<<endl;
 do
{
	getline(Fin, Str);
	cout<<Str<<endl;
}while(!Fin.eof());
Fin.close();
}
 else
{
cerr<< "Помилка відкриття файлу";
}
}
```

```
Файл открыт для записи
Введите слово :  111111111

 Будете продолжать? Нажмите 1 , усли да  1

Введите слово :  22222222222

 Будете продолжать? Нажмите 1 , усли да  1

Введите слово :  33333333333

 Будете продолжать? Нажмите 1 , усли да  я

Файл открыт для чтения
111111111
22222222222
33333333333
```

## 3. РОБОТА З БІНАРНИМИ ФАЙЛАМИ.

У С++ немає типу byte, а замість типу byte використовують тип char. Відповідно до правил мови С++, тип char виявляється найменшою одиницею інформації. Через це сприйняття нами ситуації загалом трохи спотворюється, адже ми бачимо char*, а насправді це як byte*.

У наших комп'ютерах використовується така модель пам'яті, де один осередок займає 1 байт. Залежно від ситуації тип char можна трактувати або як символьний тип, або як байтовий тип. Файли, що зберігають послідовність байтів, називають бінарними файлами. Це з тим, що байт ділиться на біти, а біти є двійкові цифри. Слово бінарний перекладається як бінарний.

Вся інформація зберігається у комп'ютері у вигляді 0 та 1, тобто у двійковому вигляді. Двійкові файли відрізняються від текстових лише методами роботи з ними. Наприклад, якщо ми записуємо в текстовий файл цифру «4», вона записується як символ, і для її зберігання потрібен один байт. Відповідно і розмір файлу дорівнюватиме одному байту. Текстовий файл, що містить запис: "145687", матиме розмір шість байт.

Якщо ж записати ціле число 145687 в двійковий файл, то він матиме розмір чотири байта, так як саме стільки необхідно для зберігання даних типу int. Тобто двійкові файли компактніші і в деяких випадках більш зручні для обробки.

Використання двійкового режиму файлу призводить до того, що дані пересилаються з пам'яті у файл або, навпаки, без перетворення. Щоб зберігати дані у двійковому форматі, при створенні потокового об'єкта (або під час відкриття) необхідно вказати режим ios::binary.

На відміну від текстового режиму цей режим повинен бути заданий явно. При явному вказівці режиму потрібно визначити всі режими відкриття файлу, поєднавши їх порозрядною операцією | (або).

```
ifstream fin(nameF, ios::in | ios::binary);
ofstream fout(nameF,ios::app |ios::binary);
fstream finout(nameF, ios::in | ios::out | ios::binary);
```

Режими відкриття файлів визначають характер використання файлів. Для встановлення режиму передбачено константи, які визначають режим відкриття файлів.

Режими відкриття файлів:

| Константа | Опис |
|---|---|
| **ios::in** | відкрити файл для читання |
| **ios::out** | відкрити файл для запису |
| **ios::ate** | при відкритті перемістити вказівник у кінець файлу |
| **ios::app** | відкрити файл для запису в кінець файлу |
| **ios::trunc** | видалити вміст файлу, якщо він існує |
| **ios::binary** | відкриття файлу в двійковому режимі |

| Поток | Предназначение | Открывается в режиме |
|---|---|---|
| ifstream | чтение файла | in |
| ofstream | запись в файл | out |
| fstream | и чтение, и запись | out \| in |

Режими відкриття файлів можна встановлювати безпосередньо під час створення об'єкта або виклику функції open().

```
ofstream fout("cppstudio.txt", ios_base::app);
// відкриваємо файл для додавання інформації до кінця файлу
fout.open("cppstudio.txt", ios_base::app);
// відкриваємо файл для додавання інформації до кінця файлу
```

Режими відкриття файлів можна комбінувати за допомогою порозрядної логічної операції або |, наприклад:

ios_base::out | ios_base::trunc – відкриття файлу для запису, попередньо очистивши його.

Об'єкти класу ofstream, при зв'язці з файлами за замовчуванням містять режими відкриття файлів:

ios_base::out | ios_base::trunc.

Тобто файл буде створено, якщо немає. Якщо ж файл існує, його вміст буде видалено, а сам файл буде готовий до запису. Об'єкти класу ifstream, зв'язуючись з файлом, мають за промовчанням режим відкриття файлу ios_base::in — файл відкритий тільки для читання. Режим відкриття файлу ще називають прапорцем.

**Запис стандартних типів даних у двійкові файли**

Щоб відкрити двійковий файл, необхідно встановити режим доступу:

ios::binary (у деяких компіляторах C++ - ios::bin).

Для створення вихідного файлу створюють об'єкт:

ofstream fout("out.bin", ios::out | ios::binary);

Запис даних відбувається за допомогою методу write(), який має два параметри:

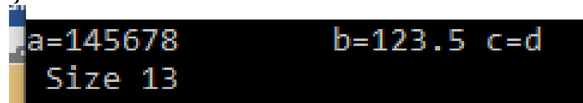
```
fout.write((char*)&X,sizeof(X));
```

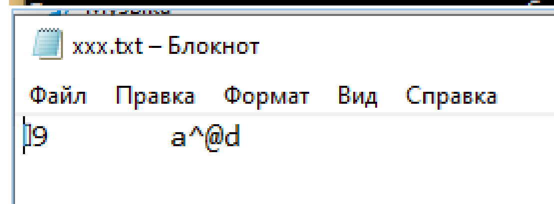
перший - покажчик на початок (адреса початку) даних, що записуються,
другий - кількість байтів, що записуються.
При цьому вказівник необхідно явно перетворити на тип char.

Цей метод копіює кілька байтів з пам'яті в файл. Кількість байтів, яке має бути скопійоване, визначається другим параметром. Перший параметр визначає адресу, де розташовані дані, які потрібно скопіювати. Особливістю методу є те, що адреса має бути перетворена на тип «покажчик на char».

**Приклад_3.1.**

```
// Запис в двійковий файл:
#include <iostream>
#include <fstream>
#include<math.h>

Void main()
{
	inta = 145 678;
	doubleb = 123.5;
	charc ='d';
	std::ofstreamFout;
	Fout.open("xxx.txt", std::ios::out | std::ios::binary);
	If (Fout.is_open())
	{
	std::cout<< "a=" <<a<< '\t' << "b=" <<b<< '\t' << "c=" <<c<< '\n';
//Дізнаємо адресу об'єкта а і наводимо об'єкт а до однобайтового типу
//Записуємо об'єкт у відкритий нами файл
		Fout.write((char*)&a,sizeof(a));
		Fout.write((char*)&b,sizeof(b));
		Fout.write((char*)&c,sizeof(c));
		Fout.seekp(0, std::ios:: end); //Стати в кінець файлу
		std::cout<< " Size " <<Fout.tellp()<< '\n';//Отримуємо поточну позицію
		Fout.close();
	}
	else
	{
		std::cout<< "ERROR!";
	}
}
```




**Читання стандартних типів даних із двійкових файлів**

Щоб відкрити існуючий двійковий файл для читання, потрібно створити об'єкт:

ifstream inpBinFile("inp.bin", ios::in I ios::binary);

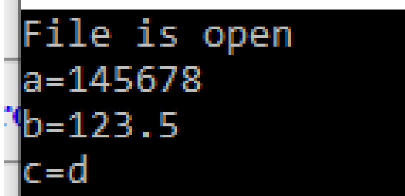
Для читання даних використовуємо функцію read(), що має аналогічні функції write() параметри.

```
fin.read ((char *) & Y, sizeof (Y));
```

Цей метод повертає значення true, якщо операція читання завершилася нормально, і false - у разі виникнення помилки, наприклад, при досягненні кінця файлу.

**Приклад_3.2.**
//Читання із двійкового файлу:

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

void main()
{

	int a;
	double b;
	char c;

	std::ifstream Fin;
	Fin.open("xxx.txt", std::ios:: in | std::ios::binary);
	if (Fin.is_open())
	{
		std:: cout<< "File is open \n";

		Fin.read((char*)&a,sizeof(a));
		Fin.read((char*)&b,sizeof(b));
		Fin.read((char*)&c,sizeof(c));
		std::cout<< "a=" <<a<< '\n';
		std::cout<< "b=" <<b<< '\n';
		std::cout<< "c=" <<c<< '\n';
		Fin.seekg(0, std::ios:: end); //Стати в кінець файлу
		std::cout<<"Size: "<<Fin.tellg()<<'\n';//Отримуємо поточну позицію
		Fin.close();
	}
	else
	{
		std::cout<< "ERROR!";
	}
}
```

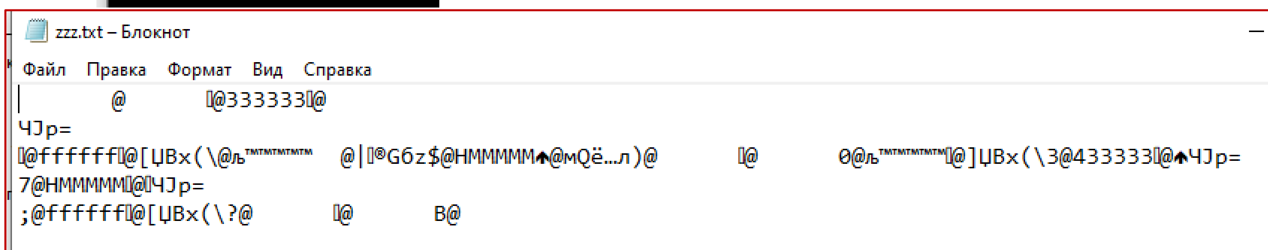



**Приклад_3.3.**
//Запис в двійковий файл результатів табулювання функції:

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

```
void main()
{
	int i;
	double a = 2, b = 6, dx;
	dx = (b – a)/10.;
	double x = 0, y = 0;

	std::ofstream Fout;
	Fout.open("zzz.txt",std::ios::out| std::ios::binary);
	if (Fout.is_open())
	{
		std::cout<< "File is open"<<std::endl;
		for(i = 0; i <= 10; i++)
		{
			x = a + i * dx;
			y = pow(x, 2);
			std::cout<<x<< '\t' <<y<<std::endl;
			Fout.write((char*)&x,sizeof(x));
			Fout.write((char*)&y,sizeof(y));
		}


		Fout.close();
	}
	else
	{
		std::cout<< "ERROR!";
	}
}
```

```
File is open
2       4
2.4     5.76
2.8     7.84
3.2     10.24
3.6     12.96
4       16
4.4     19.36
4.8     23.04
5.2     27.04
5.6     31.36
6       36
```




**Методи позиціонування.**

Оскільки при організації зовнішньої таблиці файл складається із записів однакового розміру, то легко забезпечити доступ до запису з певним номером.

Для цього використовуються методи:

**seekg()**- з об'єктами класів ifstream та fstream,

**та seekp()**- з об'єктами класів ofstream та fstream.

Довільний доступ можна однаково використовувати і для файлів, відкритих у текстовому режимі, і для файлів, відкритих у бінарному режимі.

Методи позиціонування відрізняються для вхідних та вихідних потоків: для вхідних потоків імена методів закінчуються символом 'g' (від слова get), а методи вихідних потоків закінчуються символом 'p' (від слова put).

Seekg встановити позицію читання

Seekp встановити позицію запису

Tellg дізнатися позицію курсора читання

Tellp дізнатися позицію курсора запису

Найзручніше з позиціонуванням працювати в бінарних файлах. У текстових файлах складніше відстежувати потрібну позицію, якщо, наприклад, у текстовому файлі записані цифри. Оскільки число складається з кількох цифр, скільки на яке число потрібно стрибнути, розраховувати стомливо і невигідно. У бінарному файлі більш просте для комп'ютера уявлення і нам достатньо знати кількість байт, займаних типом, щоб зробити стрибок на потрібну позицію в такому файлі і необхідно знати порядок слідування байт.

Параметри:

ios_base::beg (зсув від початку).

ios_base::cur (зміщення поточної позиції).

ios_base::end (зсув від кінця).

```
file.seekg(0,ios_base::end); //Стати в кінець файлу
file.seekg(10,ios_base::end); //Стати на 10 байтів з кінця
file.seekg(30,ios_base::beg); //Стати на 31-й байт
file.seekg(3,ios_base::cur); //перестрибнути через 3 байти
file.seekg(3); //перестрибнути через 3 байти - аналогічно
```

Версія функції seekg() з одним параметром переміщують покажчики файлів позиції, задані параметром . Це значення має бути отримано шляхом звернення або до функції tellg(), або до функції tellp() відповідно.

**Метод tellg()**

Іноді потрібно отримувати інформацію про те, скільки вже прочитано. У цьому допоможе метод tellg():

```
1    cout << "Вважають байт: " << file.tellg();
```

Він повертає значення типу int, яке показує, скільки вже пройдено в байтах. Його можна використовувати в парі з способом seekg(), щоб отримувати розмір файлу:

```
1    //стаємо наприкінці файлу
2    file.seekg(0,ios_base::end);
3    //Отримуємо поточну позицію
4    cout << "Розмір файлу (в байтах):" << file.tellg();
```

**Метод eof()**

Перевіряє, чи не кінець файлу досягнуто.

## 4. СТРУКТУРИ.

Мова C++ дозволяє програмістам створювати власні власні типи даних — типи, які групують кілька окремих змінних разом. Одним з найпростіших типів даних є структура. Структура дозволяє згрупувати змінні різних типів у єдине ціле.

Структура – це тип, який визначає користувач, застосовуючи ключове слово struct.

```
struct имя_структуры
{
   компоненти_структури
};
```

*Ім'я_структури*представляє довільний ідентифікатор, якого застосовуються самі правила, як і за найменуванні змінних.

Після імені структури у фігурних дужках поміщаються*Компоненти_структури*, які представляють набір описів об'єктів та функцій, що складають структуру.

Наприклад:

```
struct Gruppa
{
 String name;
 String surname;
 Int date;
 Int month;
 Int year;

};
```

Кожен рядок, що визначає окремий елемент структури, обмежена точкою з комою, і точка з комою з'являється після закриває дужки.

Елементи структури можуть бути будь-якого типу, крім самого типу структури, всередині якої вони визначені.

**Ініціалізація структури**

Перший спосіб встановити дані члени структури -це визначити їх початкові значення в оголошенні.

Ініціалізують значення з'являються між дужками і розділені комами - майже так само. як визначаються початкові значення елементів масиву

Послідовність ініціалізують значень очевидно має збігатися з послідовністю полів структури у її визначенні.

```
Gruppa student =
{
         "Настя",
         "Козлова",
        23,
        07,
        2004
};
```

Ініціалізацію можна зробити при оголошенні структури:

```
struct Gruppa
{

         String name="Настя";
         String surname="Козлова";
         Int date=23;
         Int month=7;
```

```
        Int year=2004;
};
```

**Доступ до членів структури**

Щоб звернутися до індивідуальних членів структури, можна використовувати операцію вибору члена (операцію доступу до члена), яка позначається точкою.

age = 2021 -student.year;

**Приклад_4.1.**

```
#include <iostream>
#include <fstream>
#include <string>
using namespacestd;
struct Gruppa
{
        String name;
        String surname;
        Int date;
        Int month;
        Int year;
};
intmain()
{
setlocale(LC_ALL,"Український");

 Gruppa student =
{       "Настя",
        "Козлова",
        23,
        07,
        2004
};
 intage;
age = 2021 – student.year;
cout<<age<<endl;
 return0;
}
```

**Масив структур:**

**Приклад_4.2.**

```
#include <iostream>
#include <fstream>
#include <string>
using namespacestd;
struct Gruppa
{
        String name="Настя";
        String surname="Козлова";
        Int date=23;
        Int month=7;
        Int year=2004;
```

```
};
intmain()
{
setlocale(LC_ALL,"Український");
 fstream inOut;
inOut.open("file.txt",ios::out);
 Gruppa student[3] =
{
{"Настя","Козлова",23,7,2004},
{"Ілля","Мельник",15,03,2004},
{"Філіп","Миронюк",20,11,2004},
};
 Int age,i;
 For (i=0;i<3;i++)
{
           = 2021 – student[i].year;

inOut<<student[i].name<< '\t' <<student[i].surname<< '\t'
 <<student[i].date<< '\t' <<student[i].month<< '\t'
 <<student[i].year<< '\t' <<age<<endl;
cout<<student[i].name<< '\t'<<student[i].surname<< '\t'
 <<student[i].date<< '\t' <<student[i].month<< '\t'
<<student[i].year<< '\t' <<age<<endl;
}
inOut.close();
cout<<endl;

inOut.open("file.txt",ios:: in);
 string Str;
cout<< "Список групи: \n";
 for(i = 0; i < 3; i++)
{
          getline(inOut, Str);
          cout<<Str<<'\n';
}
inOut.close();
 return0;
}
```

```
C:\WINDOWS\system32\cmd.exe
Настя    Козлова 23      7       2004    17
Илья     Мельник 15      3       2004    17
Филипп   Миронюк 20      11      2004    17

Список группы:
Настя    Козлова 23      7       2004    17
Илья     Мельник 15      3       2004    17
Филипп   Миронюк 20      11      2004    17
Для продолжения нажмите любую клавишу . . .
```

**Приклад_4.3.**

```
//Запис у файл та читання записів оформлені у вигляді функцій.
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct Gruppa
{
	String name;
	String surname;
	Int date;
	Int month;
	Int year;
};

void Fout(fstream&inOut);
void Fin(fstream&inOut);

int main()

setlocale(LC_ALL,"Український");

 fstream inOut;
inOut.open("file.txt",ios::out);
 if (inOut.is_open())
{
	Fout(inOut);
}
 else
{
	cout<< "ERROR!";
}

inOut.open("file.txt",ios:: in);
 if (inOut.is_open())
{
	Fin(inOut);
}
 else
{
	cout<< "ERROR!";
}
  return0;
}

Void Fout(fstream&inOut)
{
  Gruppa student[3] =
{
	{"Настя","Козлова",23,7,2004},
```

```
        {"Ілля","Мельник",15,03,2004},
        {"Філіп","Миронюк",20,11,2004},
};

 intage, i;
 for(i = 0; i < 3; i++)
{
age = 2021 – student[i].year;

 inOut <<student[i].name<< '\t' <<student[i].surname<< '\t'
 <<student[i].date<< '\t' <<student[i].month<< '\t'
 <<student[i].year<< '\t' <<age<<endl;
cout<<student[i].name<< '\t' <<student[i].surname<< '\t'
 <<student[i].date<< '\t' <<student[i].month<< '\t'
<<student[i].year<< '\t' <<age<<endl;
}
 inOut. close ();
cout<<endl;

}

void Fin(fstream&inOut)
{
 string Str;
cout<< "Список групи: \n";
 for (inti = 0; i < 3; i++)
{
        getline(inOut, Str);
        cout<<Str<< '\n';
}
 inOut. close ();
}
```

**5. КЛАСИ.**

Клас – це специфікація типу даних, визначеного вами. Він може містити елементи даних, які можуть бути змінними базових типів С++, так і інших певних користувачем типів. Елементи даних класу можуть бути окремими елементами даних, масивами, покажчиками, масивами покажчиків майже будь-якого роду, об'єктами інших класів, так що у вашому розпорядженні практично необмежену гнучкість щодо того, що можна включати у ваші типи даних. Клас також може містити функції, що оперують об'єктами класу, звертаючись до елементів даних, які вони включають. Таким чином, клас комбінує визначення елементарних даних, з яких складається об'єкт, та засоби маніпулювання даними, що належать до індивідуальних об'єктів класу.

Так само, як оголошення структури, так і оголошення класу не призводить до виділення будь-якої пам'яті. Для використання класу необхідно оголосити змінну цього типу класу.

У С++ змінна класу називається екземпляром чи об'єктом класу. Точно так, як визначення змінної фундаментального типу даних (наприклад, int x) призводить до виділення пам'яті для цієї змінної, так само і створення об'єкта класу) призводить до виділення пам'яті для цього об'єкта.

Крім зберігання даних, які називають полями, класи також можуть містити функції. Функції, визначені всередині класу, називаються функціями-членами чи методами. Методи можуть бути визначені усередині або поза класом.

Так само, як до членів структури, так і до членів (змінних та функцій) класу доступ здійснюється через оператор вибору членів (.).

**Визначення класу**

І м ' я   к л а с у слідує за ключовим словом CLASS, і між фігурними дужками визначені поля.

Поля визначені в класі з використанням операторів оголошення, а все визначення класу завершується крапкою з комою. Імена всіх членів класу локальні стосовно нього. Тому ви можете використовувати ті ж імена ще десь у програмі, і це не викликає жодних проблем.

Приклад класу **Student**:

```
class Student //клас
{
public: // Специфікатор доступу
        string name;// властивості класу - поля класу
        int age;
        int weight;
};
```

**Визначення об'єктів класу.**

Об'єкти класу оголошуютьсятаким же чином, якта об'єкти базових типів.

```
Student firstStudent;       // Об'єкт класу
Student secondStudent;      //Другий об'єкт класу Student
Student thirdStudent;       //Третій об'єкт класу Student
```

**Доступ до полів класів.**

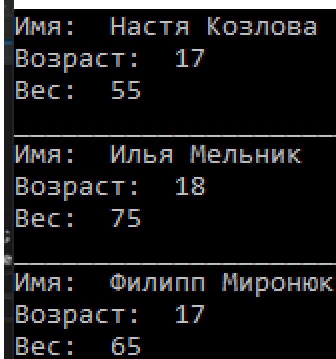
Здійснюється з використанням оператора вибору (.).

```
firstStudent.name= "Настя Козлова";
firstStudent.age = 17;
firstStudent.weight = 55;
```

**Приклад_5.1.**

```
#include <iostream>
#include <string>
using namespace std;
class Student   //клас
{
public: // Специфікатор доступу
        string name;// властивості класу - поля класу
        int age;
        int weight;
};
int main()
{
        setlocale(LC_ALL,"Український");
```

```
        Student firstStudent;  // Об'єкт класу
        firstStudent.name= "Настя Козлова";
        firstStudent.age = 17;
        firstStudent.weight = 55;

        cout<< "Ім'я:" <<firstStudent.name<<endl;
        cout<< "Вік:" <<firstStudent.age<<endl;
        cout<< "Вага: " <<firstStudent.weight<<endl;

        cout<< "_____________________________________________" <<endl;
        Student secondStudent;        //Другий об'єкт класу Student
        secondStudent.name= "Ілля Мельник";
        secondStudent.age = 18;
        secondStudent.weight = 75;

        cout<< "Ім'я:" <<secondStudent.name<<endl;
        cout<< "Вік:" <<secondStudent.age<<endl;
        cout<< "Вага: " <<secondStudent.weight<<endl;

        cout<< "_____________________________________________" <<endl;
        Student thirdStudent;         //Третій об'єкт класу Student
        thirdStudent.name= "Філіп Миронюк";
        thirdStudent.age = 17;
        thirdStudent.weight = 65;

        cout<< "Ім'я:" <<thirdStudent.name<<endl;
        cout<< "Вік:" <<thirdStudent.age<<endl;
        cout<< "Вага: " <<thirdStudent.weight<<endl;
        return0;
}
```




**Методи класів.**

Метод класу – це функція, визначення чи прототип якої усередині визначення класу. Вона оперує будь-яким об'єктом класу, членом якого є і має доступ до всіх членів класу цього об'єкта.

**Приклад_5.2.**

```
#include <iostream>
#include <string>
using namespacestd;
class Student   //клас
```

```
{
public:         // Специфікатор доступу
        string name;// властивості класу - поля класу
        int age;
        int weight;

        void print()
        {
                cout<< "Ім'я:" <<name<<endl;
                cout<< "Вік:" <<age<<endl;
                cout<< "Вага: " <<weight<<endl;
                cout<< "________________________________________" <<endl;
        }
};
int main()
{
        setlocale(LC_ALL,"Український");

        Student firstStudent;  // Об'єкт класу
        firstStudent.name= "Настя Козлова";
        firstStudent.age = 17;
        firstStudent.weight = 55;
        firstStudent.print();


        Student secondStudent;        //Другий об'єкт класу Student
        secondStudent.name= "Ілля Мельник";
        secondStudent.age = 18;
        secondStudent.weight = 75;
        secondStudent.print();

        Student thirdStudent;         //Третій об'єкт класу Student
        thirdStudent.name= "Філіп Миронюк";
        thirdStudent.age = 17;
        thirdStudent.weight = 65;
        thirdStudent.print();

        return0;
}
```

```
Имя:  Настя Козлова
Возраст:  17
Вес:  55
________________________________________
Имя:  Илья Мельник
Возраст:  18
Вес:  75
________________________________________
Имя:  Филипп Миронюк
Возраст:  17
Вес:  65
________________________________________
Для продолжения нажмите любую клавишу . . .
```

**Розташування визначення методу.**

Визначення методу має бути поміщене всередину визначення класу. Якщо ви хочете розмістити його поза визначенням класу, то всередині класу потрібно вказати його прототип.

**Приклад_5.3.**

```
#include <iostream>
#include <string>
using namespace std;
class Student //клас
{
public:           // Специфікатор доступу
          string name;    // властивості класу - поля класу
          int age;
          int weight;
          void print();
};
void Student::print()
{
          cout<< "Ім'я:" <<name<<endl;
          cout<< "Вік:" <<age<<endl;
          cout<< "Вага: " <<weight<<endl;
          cout << "_________________________________________" <<endl;
}
```

## 6. УПРАВЛІННЯ ДОСТУПОМ.

**Специфікатор доступу** визначає, хто має доступом до членів цього специфікатора. Кожен із членів «набуває» рівня доступу відповідно до специфікатора доступу (або, якщо він не вказаний, відповідно до специфікатора доступу за замовчуванням).

**У мові С++ є 3 рівні доступу**:

специфікатор **public** робить члени відкритими;

специфікатор **private** робить члени закритими; задаєтьсяза замовчуванням. Доступ усередині класу та для дружніх класів.

специфікатор **protected** відкриває доступ до членів лише для дружніх та дочірніх класів.

Класи можуть використовувати відразу кілька специфікаторів доступу для встановлення рівнів доступу для кожного із своїх членів. Зазвичай перемінні члени є закритими, а методи - відкритими.

**Правило:**

**Встановлюйте специфікатор доступу:**

**private** змінним-членам класу

**public**- методам класу.

Приклад використання специфікаторів доступу:

**Приклад_6.1.**

```
#include <iostream>
#include <string>
```

```
using namespace std;
class Student //клас
{
public:// Специфікатор доступу
        string name;// властивості класу - поля класу
        int age;
        int weight;
        void print();

private:
        int weight;
};

void Student::print()
{
        cout<< "Ім'я:" <<name<<endl;
        cout<< "Вік:" <<age<<endl;
        cout<< "Вага: " <<weight<<endl;
        cout<< "________________________________________" <<endl;
}

intmain()
{
        setlocale(LC_ALL,"Український");

        Student firstStudent;           // Об'єкт класу
        firstStudent.name= "Настя Козлова";
        firstStudent.age = 17;
        firstStudent.weight = 55;    // Змінна недоступна!
        firstStudent.print();
return0;
}
```

**Приклад_6.2.**

```
#include <iostream>
#include <string>
using namespace std;
class Student   //клас
{
public:          // Специфікатор доступу
        string name;    // властивості класу - поля класу
        int age;
        void print();   // функція бачить усі поля класу

private:
        int weight;
};

void Student::print()
{
        weight = 55;
        cout<< "Ім'я:" <<name<<endl;
```

```
        cout<< "Вік:" <<age<<endl;
        cout<< "Вага: " <<weight<<endl;
        cout<< "________________________________________" <<endl;
}

Int main()
{
        setlocale(LC_ALL,"Український");

        Student firstStudent;           // Об'єкт класу
        firstStudent.name= "Настя Козлова";
        firstStudent.age = 17;
        //firstStudent.weight = 55;
        firstStudent.print();
return0;
}
```

**Інкапсуляція**

В об'єктно-орієнтованому програмуванні інкапсуляція (або ще приховування інформації) - це процес прихованого зберігання деталей реалізації об'єкта. Користувачі звертаються до об'єкта через відкритий інтерфейс.

У C++ інкапсуляція реалізована через **специфікатори доступу**. Зазвичай, всі змінні-члени класу є закритими (приховуючи деталі реалізації), більшість методів є відкритими (з відкритим інтерфейсом користувача).

Інкапсуляція має багато переваг, основна з яких це використовувати клас без необхідності розуміння його реалізації. Інкапсульовані класи:

простіше у використанні;

зменшують складність програми;

захищають та запобігають дані від неправильного використання;

легше змінювати;

легше налагоджувати.

Це значно полегшує використання класів.

**Функції доступу (гетери та сеттери).**

Залежно від класу може бути доречним (в контексті того, що робить клас) мати можливість отримувати/встановлювати значення закритим змінним-членам класу.

**Функція доступу**— це коротка відкрита функція, завданням якої є отримання чи зміна значення закритої змінної-члена класу.

Наприклад:

**Функції доступу зазвичай бувають двох типів**:

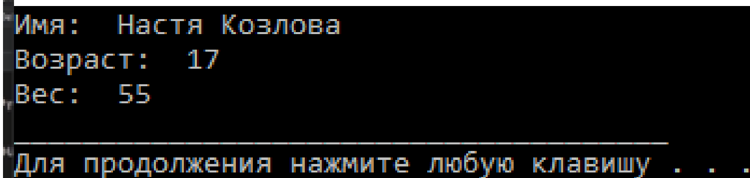
**гетери** - це функції, які повертають значення закритих змінних членів класу;

**сеттери** - це функції, які дозволяють надавати значення закритим змінним членам класу.

**Приклад_6.3.**

```
#include <iostream>
#include <string>
using namespace std;
class Student //клас
{
private :        // Специфікатор доступу
```

```
        string name;    // властивості класу - поля класу
        int age;
        int weight;
public:
        void SetName (string VoidName)
        {
                name= VoidName;
        }
        Void SetAge (int VoidAge)
        {
                age =VoidAge;
        }
        Void SetWeight (int VoidWeight)
        {
                weight =VoidWeight;
        }

        Void print();// функція бачить усі поля класу

};

void Student::print()
{
        cout<< "Ім'я:" <<name<<endl;
        cout<< "Вік:" <<age<<endl;
        cout<< "Вага: " <<weight<<endl;
        cout<< "________________________________________" <<endl;
}
intmain()
{
        setlocale(LC_ALL,"Український");
        Student firstStudent;  // Об'єкт класу
        firstStudent.SetName("Настя Козлова");
        firstStudent.SetAge(17);
        firstStudent.SetWeight(55);
        firstStudent.print();
                return0;
}
```




**Приклад_6.4.**

```
#include <iostream>
#include <string>
using namespace std;
class Student   //клас
{
```

```
private:        //Модифікатор доступу
        string name;    // властивості класу - поля класу
        int age;
        int weight;
public:
        void SetName (string VoidName)// сетер
        {
                name= VoidName;
        }
        void SetAge (int VoidAge)
        {
                age =VoidAge;
        }
        void SetWeight (int VoidWeight)
        {
                weight =VoidWeight;
        }
        int GetWeight()         // Геттер
        {
                returnweight;
        }
        void print();   // функція бачить усі поля класу
        };
void Student::print()
{
        cout<< "Ім'я:" <<name<<endl;
        cout<< "Вік:" <<age<<endl;
        cout<< "Вага: " <<weight<<endl;
        cout<< "______________________________________________" <<endl;
}

intmain()
{
        setlocale(LC_ALL,"Український");

        Student firstStudent;           // Об'єкт класу
        firstStudent.SetName("Настя Козлова");
        firstStudent.SetAge(17);
        firstStudent.SetWeight(55);
        firstStudent.print();
        intnewWeight;
        newWeight = firstStudent.GetWeight()+10;
        cout<< "Нова вага:" <<newWeight<<endl;
        return0;
}
```

```
C:\WINDOWS\system32\cmd.exe
Имя:  Настя Козлова
Возраст:  17
Вес:  55
______________________________________
Новый вес: 65
Для продолжения нажмите любую клавишу . . .
```

## 7. КОНСТРУКТОР КЛАСУ.

Конструктор класу – це спеціальна функція класу, що викликається під час створення нового об'єкта цього класу. Таким чином, вона надає можливість ініціалізувати об'єкти під час їх створення та гарантувати, що всі поля матимуть коректні значення. Клас може мати кілька конструкторів, дозволяючи створювати об'єкти у різний спосіб.

Конструктори завжди називаються на ім'я класу, в якому визначені.

Конструктор немає типу повернення. Основне призначення конструктора класу – надавати початкові значення елементам даних класу, і жодного типу повернення не потрібно і не допускається.

**Приклад_7.1.**
// Використання конструктора:

```
#include <iostream>
#include <string>
using namespace std;
class Student //клас
{
private:// Специфікатор доступу
	string name;// властивості класу - поля класу
	int age;
	int weight;
public:
	Student(string nameS,int ageS,int weightS)
	{
		name= nameS;
		age =ageS;
		weight =weightS;
	}

		int GetWeight()// Геттер
	{
		return weight;
	}
	void print();// функція бачить усі поля класу
};

void Student::print()
{
	cout<< "Ім'я:" <<name<<endl;
```

```
        cout<< "Вік:" <<age<<endl;
        cout<< "Вага: " <<weight<<endl;
        cout<< "________________________________________" <<endl;
}

int main()
{
        setlocale (LC_ALL,"Український");

Student firstStudent ("Настя Козлова",17,55);//Оголошення та
                                            //ініціалізація об'єкта класу
        firstStudent.print();

        int newWeight;
        newWeight = firstStudent.GetWeight() + 10;
        cout<< "Нова вага:" <<newWeight<<endl;
        return0;
}
```

```
Имя:  Настя Козлова
Возраст:  17
Вес:  55
________________________________________
Новый вес: 65
Для продолжения нажмите любую клавишу . . .
```

**Конструктор за замовчуванням.**

Конструктор, який не має параметрів (або має параметри, всі з яких маютьзначення за замовчуванням), називається конструктором за замовчуванням. Він викликається, якщо користувач не вказує значення для ініціалізації.

```
//Визначення конструктора за умовчанням
Student() { }

//Оголошення об'єкта без ініціалізації
Student firstStudent;
```

**Приклад_7.2.**

```
#include <iostream>
#include <string>
using namespace std;
class Student //клас
{
private:        // Специфікатор доступу
        string name;    // властивості класу - поля класу
        int age;
        int weight;
public:
        Student() //Визначення конструктора за замовчуванням
        {
                name= "Настя Козлова";
```

```
            age = 17;
            weight = 55;
      }

      Student (string nameS,int ageS,int weightS)
      {
            name= nameS;
            age =ageS;
            weight =weightS;
      }
      cout<< "Ім'я:" <<name<<endl;
      cout<< "Вік:" <<age<<endl;
      cout<< "Вага: " <<weight<<endl;
      cout<< "_______________________________________" <<endl;
}

int main()
{
      Setlocale (LC_ALL,"Український");

      Student firstStudent; // Об'єкт класу без ініціалізації
            firstStudent.print();


      StudentsecondStudent("Ілля Мельник",18,75);//Другий об'єкт
            secondStudent.print();

      Student thirdStudent("Філіп Миронюк",17,65);//Третій об'єкт
            thirdStudent.print();

      return0;
}
```

```
Имя:  Настя Козлова
Возраст:  17
Вес:  55
_______________________________________
Имя:  Илья Мельник
Возраст:  18
Вес:  75
_______________________________________
Имя:  Филипп Миронюк
Возраст:  17
Вес:  65
_______________________________________
Для продолжения нажмите любую клавишу . . .
```

**Надання параметрів у класі значення за замовчуванням**

Можна визначити значення параметрів за промовчанням у прототипі методу класу,включаючиконструктори. Якщо ви помістите визначення методу всередину визначення класу, то можете вказати значення її параметрів за промовчанням у заголовку функції. Якщо ж визначення класу включений лише прототип функції, то цьому прототипі повинні бути значення параметрів за промовчанням.

**Приклад_7.3.**
//Використання значень за промовчанням для аргументів конструктора:

```
#include <iostream>
#include <string>
using namespace std;
class Student //клас
{
private://Модифікатор доступу
	string name;// властивості класу - поля класу
	int age;
	int weight;
public:
	Student(string nameS="Настя Козлова",int ageS=17,int weightS=55)
	{
		name= nameS;
		age =ageS;
		weight =weightS;
	}
	int GetWeight()// Геттер
	{
		return weight;
	}
	void print();// функція бачить усі поля класу

};
void Student::print()
{	cout<< "Ім'я:" <<name<<endl;
	cout<< "Вік:" <<age<<endl;
	cout<< "Вага: " <<weight<<endl;
	cout<< "_____________________________________________" <<endl;
}
int main()
{
	setlocale(LC_ALL,"Український");

	Student firstStudent;// Перший об'єкт класу

	firstStudent.print();

	Student secondStudent("Ілля Мельник",18,75);//Другий об'єкт класу Student

	secondStudent.print();

	Student thirdStudent("Філіп Миронюк",17,65);//Третій об'єкт класу Student

	thirdStudent.print();

	return0;
}
```

```
Имя:  Настя Козлова
Возраст:  17
Вес:  55
_______________________________________
Имя:  Илья Мельник
Возраст:  18
Вес:  75
_______________________________________
Имя:  Филипп Миронюк
Возраст:  17
Вес:  65
_______________________________________
Для продолжения нажмите любую клавишу . . .
```

## 8. ДЕСТРУКТОР КЛАСУ. КЛЮЧОВЕ СЛОВО *THIS.*

Деструктор це ще один спеціальний тип методу класу, який виконується при видаленні об'єкта класу. Коли конструктори призначені для ініціалізації класу, деструктори призначені для очищення пам'яті після нього.

Коли об'єкт автоматично виходить з області видимості або динамічно виділений об'єкт явно видаляється за допомогою ключового слова delete, Викликається деструктор класу (якщо він існує) для виконання необхідної очищення до того, як об'єкт буде видалений з пам'яті. Для простих класів (тих, які ініціалізують значення звичайних змінних-членів) деструктор не потрібен, оскільки C++ автоматично виконає очищення самостійно.

Однак, якщо об'єкт класу містить будь-які ресурси (наприклад, динамічно виділену пам'ять або файл/базу даних), або якщо вам необхідно виконати будь-які дії до того, як об'єкт буде знищений, деструктор є ідеальним рішенням, оскільки він останнє, що відбувається з об'єктом перед його остаточним знищенням.

Так само, як і конструктори, деструктори мають свої правила, що стосуються їхніх імен:

— Деструктор повинен мати те саме ім'я, що й клас зі знаком тильда (~) на самому початку.

— Деструктор не може ухвалювати аргументи.

- Деструктор не має типу повернення.

З другого правила випливає ще одне правило: для кожного класу може існувати тільки один деструктор, тому що можливості перевантажити деструктори як функції - немає і відрізнятися один від одного аргументами вони не можуть.

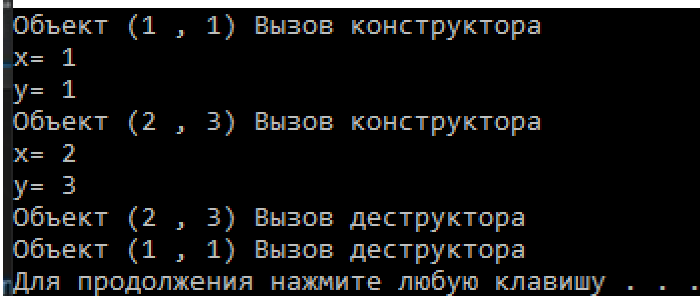
Подібно до конструкторів, деструктори не викликаються явно. Однак їх можуть безпечно викликати інші методи класу, тому що об'єкт не знищиться доти, доки не виконається деструктор.

**Приклад 8.1**.

// Реалізація деструктора:

```
#include <iostream>
#include <string>
using namespace std;
class Point
{
private:
	double m_x;
	double m_y;
```

```
public:
        Point (double x,double y)
        {
        cout<<"Об'єкт ("<<x<<" , "<<y<< ") Виклик конструктора" <<endl;
                m_x =x;
                m_y =y;
        }
~Point()
{
cout<< "Об'єкт (" <<m_x<< " , " <<m_y<< ") Виклик деструктора" <<endl;
}
        void print()
        {
                cout<< "x=" <<m_x<<endl;
                cout<< "y=" <<m_y<<endl;
        }
};

intmain()
{
        setlocale(LC_ALL,"Український");

        Point a (1., 1.);
        a.print();
        Point b (2., 3.);
        b.print();

        return0;
}
```



```
Объект (1 , 1) Вызов конструктора
x= 1
y= 1
Объект (2 , 3) Вызов конструктора
x= 2
y= 3
Объект (2 , 3) Вызов деструктора
Объект (1 , 1) Вызов деструктора
Для продолжения нажмите любую клавишу . . .
```

**Приклад_8.2.**

// Реалізація деструктора:

```
#include <iostream>
#include <string>
using namespace std;
class Point
{
private:
        doublem_x;
        doublem_y;
public:
        Point(double x,double y)
```

```
	{
		cout<<"Об'єкт ("<<x<<" , "<<y<< ") Виклик конструктора" <<endl;
		m_x =x;
		m_y =y;
	}
	~Point()
	{
cout<< "Об'єкт (" <<m_x<< " , " <<m_y<< ") Виклик деструктора" <<endl;
	}
	void print()
	{
		cout<< "x=" <<m_x<<endl;
		cout<< "y=" <<m_y<<endl;
	}
};
void Func()
{
	cout<< "Початок виконання" <<endl;
		Pointa(1., 1.);
		a.print();
	cout<< "Кінець виконання" <<endl;
}
int main()
{
	setlocale(LC_ALL,"Український");
	Func();
	Point b(2., 3.);
	b.print();
	return0;
}
```

```
Начало выполнения
Объект (1 , 1) Вызов конструктора
x= 1
y= 1
Конец выполнения
Объект (1 , 1) Вызов деструктора
Объект (2 , 3) Вызов конструктора
x= 2
y= 3
Объект (2 , 3) Вызов деструктора
Для продолжения нажмите любую клавишу . . .
```

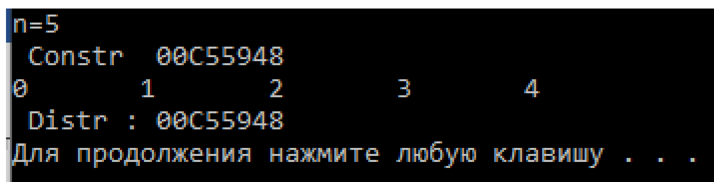
**Деструктори та динамічний розподіл пам'яті**

Нерідко виникає потреба у динамічному виділенні пам'яті для полів класу. Ви можете скористатися операцією new у конструкторі, щоб виділити місце у пам'яті для поля об'єкта. У такому разі відповідальність за звільнення цього місця в пам'яті, коли об'єкт не потрібен, повинна бути покладена на відповідний деструктор.

**Приклад_8.3.**

//Реалізація деструктора:

```
#include <iostream>
using namespace std;
class Array
{
private:
 int* m_array;
public:
Array(int size)
{
m_array =new int[size];
 for(int i = 0; i <size; i++)
{
	m_array[i] = i;
}
cout<< "Constr" <<m_array<<endl;
}
~Array()
{
 delete[]m_array;
cout<< " Distr : " <<m_array<<endl;
}

void print(int size)
{
 for (int i = 0; i <size; i++)
{
	cout<<m_array[i]<< '\t';
}
cout<<endl;
}
};
int main()
{
 int n;//Розмірність масиву
cout<< "n=";
cin>>n;
 Array mas(n);
mas.print(n);
 return0;
}
```




**Ключове слово *this***

Ключове слово цієї мови програмування С++ представляє покажчик на поточний об'єкт даного класу. Таким чином, через це ми зможемо звернутися всередині класу до будь-яких членів цього класу.

**Приклад_8.4.**

//Створити клас куль, заданих координатами центру та радіусом. Створити методи, що повертають значення об'єму кулі, площу поверхні і порівнюють їх обсяги.

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

using namespace std;
class Ball
{
 double m_CentX;
 double m_CentY;
 double m_Rad;
public:
Ball(double m_CentX,double m_CentY,double Radius)
{
cout<< "Constr Radius=" << Radius<<endl;

 // m_CentX = CenterX;
 this->m_CentX =m_CentX;

// m_CentY = CenterY;
 this->m_CentY =m_CentY;

m_Rad =Radius;
}

 doublevolume()
{
 return4*3.14*pow(m_Rad, 3)/3;
};

 double area()
{
 return4*3.14*pow (m_Rad, 2);
};

 bool Compare(Ball balls2)
{

 return this->volume() >balls2.volume();
};

~Ball()
{
cout<< "Distr Radius=" <<m_Rad<<endl;
};
};

I nt main()
{
 Ball ball1(2., 3., 5.);
```

```
cout<< "V_1=" <<ball1.volume()<<endl;
cout<< "S_1=" <<ball1.area()<<endl;
 Ball ball2(1., 3., 4.);
cout<< "V_2=" <<ball2.volume()<<endl;
cout<< "S_2=" <<ball2.area()<<endl;

 if (ball1.Compare(ball2))
{
        cout<< "valume of ball1 > valume of ball2" <<endl;
}
 else
{
        cout<< "valume of ball1 < valume of ball2" <<endl;

}
 return0;
}
```

## 9. ВИКОРИСТАННЯ СПИСКУ ІНІЦІАЛІЗАЦІЇ У КОНСТРУКТОРІ. ДРУЖНІ ФУНКЦІЇ КЛАСУ.

**Використання списку ініціалізації у конструкторі**

Раніше ви ініціалізували поля об'єкта у конструкторі класу, використовуючи явні аргументи. Однак існує й інша техніка, яка ґрунтується на застосуванні списку ініціалізації.

Визначення конструктора з використанням списку ініціалізації:

**Ball():m_CentX(2.), m_CentY(3.), m_Rad(5.)**

{}

**Приклад_9.1.**

```
//Використання списку ініціалізації у конструкторі:
#include <iostream>
#include<math.h>

using namespace std;
class Ball
{
        double m_CentX;
        doubl em_CentY;
         double m_Rad;
         double m_Pi;
public:
        Ball(double CentX,double CentY,double Rad,double Pi= 3.14)
        :m_CentX(CentX), m_CentY(CentY), m_Rad(Rad), m_Pi (Pi)
        {
        cout<< "Constr Radius=" << Rad <<endl;
        }

 double volume()
```

```
{
        return4 * m_Pi * pow (m_Rad, 3) / 3;
};
 double area()
{
        return4 * m_Pi * pow (m_Rad, 2);
};

 bool Compare(Ball balls2)
{
 return this->volume() >balls2.volume();
};

~Ball()
{
cout<< "Distr Radius=" <<m_Rad<<endl;
};
};
int main()
{
 Ball ball1(2., 3., 5.);
cout<< "V_1=" <<ball1.volume()<<endl;
cout<< "S_1=" <<ball1.area()<<endl;
 Ball ball2(1., 3., 4.);
cout<< "V_2=" <<ball2.volume()<<endl;
cout<< "S_2=" <<ball2.area()<<endl;

 if(ball1.Compare(ball2))
{
        cout<< "valume of ball1 > valume of ball2" <<endl;
}
 else
{
        cout<< "valume of ball1 < valume of ball2" <<endl;

}
 return0;
}
```

```
Constr Radius= 5
V_1= 523.333
S_1= 314
Constr Radius= 4
V_2= 267.947
S_2= 200.96
Distr Radius= 4
valume of  ball1 > valume of  ball2
Distr Radius= 4
Distr Radius= 5
Для продолжения нажмите любую клавишу . . .
```

**Дружні функції класу**

Бувають випадки, коли з тієї чи іншої причини ви хочете деяким функціям, які не є членами класу, дозволити доступ до всіх членів класу, тобто визначити різновид елітної групи зі спеціальними привілеями.

Такі функції називаються дружніми функціями класу та визначаються за допомогою ключового слова ***friend***. Можна включити або тільки прототип дружньої функції визначення класу, або все визначення функції цілком.

Функції, які є друзями класу і певні всередині визначення класу, за замовчуванням є вбудованими.

Дружні функції є членами класу – це просто глобальні функції зі спеціальними привілеями. Атрибути доступу до неї не належать.

**Приклад_9.2.**

//Необхідно реалізувати дружню функцію у класі Ball для обчислення площі поверхні об'єкта Ball.

```
#include <iostream>
#include<math.h>
using namespace std;
class Ball
{
        double m_CentX;
        double m_CentY;
        double m_Rad;
        double m_Pi;

public:
        Ball(double CentX,double CentY,double Rad,double Pi= 3.14)
        :m_CentX(CentX), m_CentY(CentY), m_Rad(Rad), m_Pi (Pi)
        {
        cout<< "Constr Radius=" << Rad <<endl;
        }

 double volume()
{
        return4 * m_Pi * pow (m_Rad, 3) / 3;
}

 bool Compare(Ball balls2)
{
        return this->volume() >balls2.volume();
};
~Ball()
{
cout<< "Distr Radius=" <<m_Rad<<endl;
}
friend doublearea(Ball Fball);
};
doublearea(Ball Fball)
{
        return4*Fball.m_Pi * pow (Fball.m_Rad, 2);
}
int main()
```

```
{
 Ball ball1(2., 3., 5.);
cout<< "V_1=" <<ball1.volume()<<endl;
 cout<< "S_1=" <<area(ball1)<<endl;
 Ballball2(1., 3., 4.);
cout<< "V_2=" <<ball2.volume()<<endl;
 cout<< "S_2=" <<area(ball2)<<endl;
 if(ball1.Compare(ball2))
{
        cout<< "valume of ball1 > valume of ball2" <<endl;
}
 else
{
        cout<< "valume of ball1 < valume of ball2" <<endl;
}
 return0;
}
```

```
Constr Radius= 5
V_1= 523.333
Distr Radius= 5
S_1= 314
Constr Radius= 4
V_2= 267.947
Distr Radius= 4
S_2= 200.96
Distr Radius= 4
valume of  ball1 > valume of  ball2
Distr Radius= 4
Distr Radius= 5
Для продолжения нажмите любую клавишу . . .
```

# 10. СПАДКУВАННЯ КЛАСІВ ТА ВІРТУАЛЬНІ ФУНКЦІЇ.

Спадкування в С + + відбувається між класами і має тип відносин «є». Клас, від якого успадковують, називається батьківським (або ще «базовим» або «суперкласом»), а клас, який успадковує, називається дочірнім (або ще «похідним» або «підкласом»).

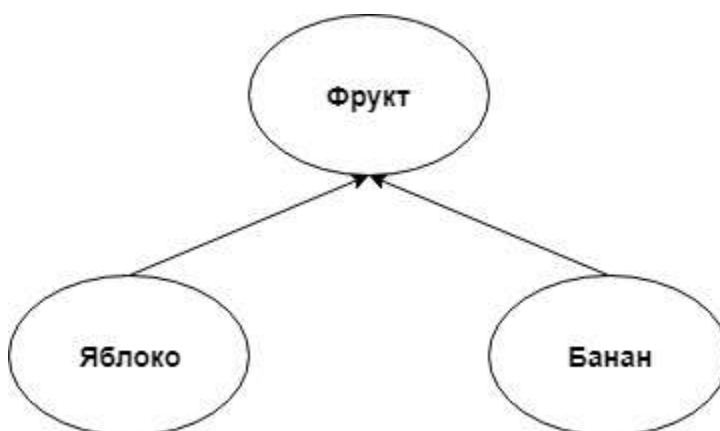



Рис. 10.1

На рис. 10.1 Фрукт є батьківським класом, а Яблуко і Банан- дочірніми.

Рис. 10.2

На рис. 10.2 Трикутник є дочірнім класом.( батько - Фігура) та батьківським (для Правильного трикутника) одночасно.

Базовий клас - це будь-який клас, який ви використовуєте як основу для визначення іншого класу.

Похідний клас автоматично отримує поля того класу, який використаний як базовий при його визначенні, і з деякими обмеженнями, також методи цього базового класу. Кажуть, що клас успадковує поля та методи класу, на якому він базується.

Єдиними членами базового класу, які не успадковуються похідним класом, є деструктор, конструктори та будь-які методи, що перевантажують операцію привласнення. Всі інші методи разом з усіма полями базового класу успадковуються похідним класом.

Ці методи та змінні стають членами дочірнього класу.

Оскільки дочірні класи є повноцінними класами, вони можуть (звичайно) мати й власні члени.

**Приклад_10.1.**

//Простий клас Human для подання Людини:

```
#include <string>
class Human
{
public:
        std::string m_name;
        int m_age;
        Human(std::string name="",int age= 0)
```

```
                    : m_name(name), m_age(age)
            {
            }
            std::string getName()
    const{return m_name; }
            int getAge()
    const{return m_age; }
    };
```

У цьому класі ми визначили ті члени, які є спільними для всіх об'єктів цього класу. Кожен Людина (незалежно від статі, професії тощо) має Ім'я і Вік.

Зверніть увагу, у прикладі вище ми зробили всі змінні-члени та методи класу відкритими. Це зроблено задля простоти прикладу. Зазвичай змінні-члени потрібно робити private.

**Наслідування класів від базового класу**

Для базового класу за замовчуванням встановлюється специфікатор private доступу, коли ви визначаєте похідний клас.

Завжди необхідно вказувати специфікатор доступу до базового класу, який визначає стан успадкованих членів у похідному класі.

Якщо ви опустите специфікатор доступу до базового класу, компілятор має на увазі private.

Зі специфікатором доступу public до базового класу всі успадковані члени, спочатку специфіковані в базовому класі як public, матимуть той самий рівень доступу в класі-спадкоємці.

Приклад:

Додамо похідний клас **Student** .

Похідний клас **Student** має одне поле, яке визначає групу студента.

Ім'я базового класу Human з'являється після імені похідного класу **Student** і відокремлюється від нього двокрапкою.

Конструктор ініціалізує змінну *m_Grup*а деструктор звільняє пам'ять. Ви також маєте передбачити в конструкторі значення за замовчуванням.

**Приклад_10.2.**

//Подивимося, як похідний клас, на відповідному прикладі.

```
#include <iostream>
#include <string>

class Human
{
public:
std::string m_Name;
 int m_year;
Human(std::string Name="",int year=2004)
:m_Name(Name), m_year(year)
{
}
std::string GetName()
{
        return(m_Name);
}
```

```
 int age()
{
        return2021 – m_year;
}
};

class Student:Human        // Специфікатор доступу private за замовчуванням
{
public:
        std::stringm_Grup;
        Student(std::string Grup ="")
        :m_Grup(Grup)
{
}
~Student()
{}
};

int main()
{
setlocale(LC_ALL,"ru");
 Student player1;
player1.m_Name= "Філіп";
std::cout<<"Ім'я:"<<player1.GetName()<<std::endl;
std::cout<<"Вік:"<<player1.age()<<std::endl;
 return0;
}
```

Компілятор видає таке повідомлення:

| | | |
|---|---|---|
| ⊗ | C2247 | нет доступа к "Human::m_Name", поскольку "Student" использует "private" для наследования из "Human" |
| ⊗ | C2247 | нет доступа к "Human::GetName", поскольку "Student" использует "private" для наследования из "Human" |
| ⊗ | C2247 | нет доступа к "Human::age", поскольку "Student" использует "private" для наследования из "Human" |

Це ясно вказує на те, що член *m_Name* з базового класу недоступний, бо *m_Name* у похідному класі став приватний. Це сталося тому, що для базового класу за замовчуванням встановлюється специфікатор доступу private, коли ви визначаєте похідний клас.

Завжди необхідно вказувати специфікатор доступу до базового класу, який визначає стан успадкованих членів у похідному класі. Якщо ви опустите специфікатор доступу до базового класу, компілятор має на увазі private. Якщо змінити визначення класу Student наступним чином:

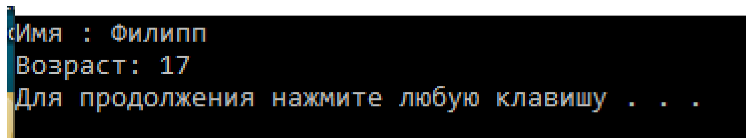
**Приклад_10.3.**

```
class Student:public Human
{
public:
std::string m_Grup;

Student (std::string Grup ="")
:m_Grup(Grup)
{
```

```
}
~Student()
{}
};

int main()
{
	setlocale(LC_ALL,"ru");
	 Student player1;
	player1.Set("Філіп");
	std::cout<<"Ім'я:"<<player1.GetName()<<std::endl;
	std::cout<<"Вік:"<<player1.age()<<std::endl;
 return0;
}
```

член m_Name буде успадкований у похідному класі як public і стане доступним функції main (). Зі специфікатором доступу public до базового класу всі успадковані члени, спочатку специфіковані в базовому класі як public, матимуть той самий рівень доступу в класі-спадкоємці.

Результат виконання програми:




**Управління доступом під час успадкування.**

Питання про доступ до успадкованих членів у похідному класі потребує більш ретельного вивчення. Розглянемо стан private-членів базового класу у похідному класі.

Хоча private-члени базового класу є також членами похідного класу, вони залишаються private по відношенню до базового класу, тому методи, додані до похідного класу, не можуть отримати доступ до них. Вони можуть бути доступні у похідному класі через методи базового класу, які не перебувають у розділі private базового класу.

**Звернення до приватних членів базового класу**

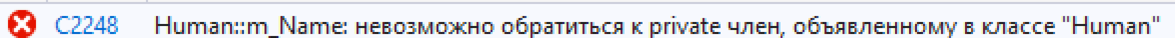
**Приклад_10.4.**

//Оголосимо поля доступу у базовому класі зі специфікатором доступу private.

```
#include <iostream>
#include <string>

class Human
{
private:
	std::string m_Name;
	int m_year;
public:
Human(std::string Name="",int year=2004)
:m_Name(Name), m_year(year)
{
```

```
}
std::string GetName()
{
        return(m_Name);
}
 int age()
{
        return 2021 – m_year;
}
 void Set(std::string Name)
{
        m_Name= Name;
}
};
class Student:public Human
{
public:
std::string m_Grup;

Student(std::string Grup ="")
:m_Grup(Grup)
{
}
~Student()
{ }
};
int main()
{
setlocale(LC_ALL,"ru");
 Student player1;
player1.m_Name= "Філіп";
std::cout<<"Ім'я:"<<player1.GetName()<<std::endl;
std::cout<<"Вік:"<<player1.age()<<std::endl;
 return 0;
}
```

Компілятор видає таке повідомлення:


C2248 Human::m_Name: невозможно обратиться к private член, объявленному в классе "Human"

Дані базового класу зі специфікатором private доступу можуть бути доступні у похідному класі через методи базового класу:

```
int main()
{
setlocale(LC_ALL,"ru");
 Student player1;
 // player1.m_Name = "Філіп";
player1.Set("Філіп");
std::cout<<"Ім'я:"<<player1.GetName()<<std::endl;
std::cout<<"Вік:"<<player1.age()<<std::endl;
```

```
 return 0;
}
```

Результат виконання програми:

```
Имя : Филипп
Возраст: 17
Для продолжения нажмите любую клавишу . . .
```

**Робота конструктора у похідному класі**

Хоча конструктори базового класу не успадковуються у похідному класі, все ж таки вони існують у базовому класі і використовуються для створення тієї частини об'єкта похідного класу, яка відноситься до базового класу. Це тим, створення цієї частини об'єкта похідного класу - справді завдання конструктора базового класу, а чи не конструктора похідного класу.

Зрештою, ви бачили, що приватні члени базового класу недоступні в об'єкті похідного класу, незважаючи на те, що вони успадковані, тому відповідальність за них лежить на конструкторах базового класу.

У наведеному нижче прикладі, конструктор базового класу за замовчуванням викликається автоматично, щоб створити частину об'єкта похідного класу, яка відноситься до базового класу.

**Приклад_10.5.**

//Конструктор базового класу за замовчуванням викликається автоматично.

```
#include <iostream>
#include <string>
class Human
{
private:
	std::string m_Name;
	int m_year;
public:
Human(std::string Name="",int year=2004)
:m_Name(Name), m_year(year)
{
	std::cout<< "Consr Human" <<std::endl;
}
std::string GetName()
{
	return(m_Name);
}
 int age()
{
 Return 2021 – m_year;
}
 Void Set(std::string Name)
{
	m_Name= Name;
}
};
class Student:public Human
{
public:
std::string m_Grup;
```

```
//Конструктор для встановлення вмісту автоматично викликає Human за замовчуванням

Student( std::string Grup ="")
{
m_Grup= Grup;
std::cout<< "Consr Student" <<std::endl;
}
~Student()
{}
};

intmain()
{
setlocale(LC_ALL,"ru");

 Humanperson1("Ілля", 2004);

std::cout<< "Human" <<std::endl;
std::cout<< "Ім'я:" <<person1.GetName()<<std::endl;
std::cout<< "Вік:" <<person1.age()<<std::endl;

 Studentplayer1("ТА2113");

player1.Set("Філіп");
std::cout<< "Student" <<std::endl;
std::cout<< "Група:" <<player1.m_Grup<<std::endl;
std::cout<<"Ім'я:"<<player1.GetName()<<std::endl;
std::cout<<"Вік:"<<player1.age()<<std::endl;
 return0;
}
```

Результати виконання програми:

```
 Consr Human
Human
Имя : Илья
Возраст: 17
 Consr Human
 Consr Student
Student
Группа : ТА2113
Имя : Филипп
Возраст: 17
Для продолжения нажмите любую клавишу . . .
```

Ви можете викликати певний конструктор базового класу конструктора похідного класу. Це дозволить ініціалізувати поля базового класу конструктором, відмінним від конструктора за замовчуванням, або вибрати певний конструктор базового класу, залежно від даних, переданих конструктору похідного класу.

**Виклик конструкторів**

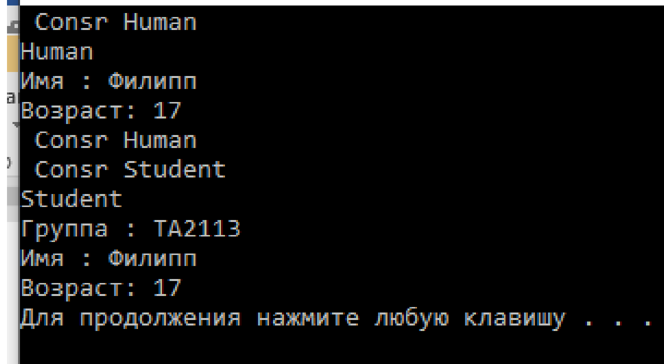
Щоб зробити похідний клас зручним у застосуванні, необхідно надати конструктор похідного класу, який дозволить визначити значення полів об'єкта. Для цього ви можете

додати додатковий конструктор у похідний клас і викликати в ньому явно конструктор базового класу, щоб встановити значення полів, успадкованих від базового класу.

**Приклад_10.6.**

```
#include <iostream>
#include <string>

class Human
{
private:
	std::string m_Name;
	int m_year;

public:
Human(std::string Name="",int year=2004)
:m_Name(Name), m_year(year)
{
	std::cout<< "Consr Human" <<std::endl;
}
std::string GetName()
{
	return(m_Name);
}
 Int age()
{
	return 2021 – m_year;
}
 void Set(std::string Name)
{
	m_Name= Name;
}
};

class Student:public Human
{
public:
std::stringm_Grup;

//Конструктор для встановлення розмірів та вмісту з явним викликом конструктора Human
Student(std::string Name,int year, std::string Груп="") :Human(Name,year)
 {
	m_Grup= Груп;
	std::cout<< "Consr Student" <<std::endl;
}
~Student()
{}
};

int main()
{
```

```
setlocale(LC_ALL,"ru");
 Human person1("Філіп", 2004);
std::cout<< "Human" <<std::endl;
std::cout<< "Ім'я:" <<person1.GetName()<<std::endl;
std::cout<< "Вік:" <<person1.age()<<std::endl;

 Student player1("Філіп",2004,"ТА2113");

std::cout<< "Student" <<std::endl;
std::cout<< "Група:" <<player1.m_Grup<<std::endl;
std::cout<<"Ім'я:"<<player1.GetName()<<std::endl;
std::cout<<"Вік:"<<player1.age()<<std::endl;
 return 0;
}
```




Зверніть увагу, що конструктор базового класу завжди викликається перед конструктором похідного класу.

Коли викликається конструктор похідного класу, конструктор базового класу завжди викликається для побудови тієї частини об'єкта похідного класу, яка належить до базового класу. Якщо ви не визначаєте використовуваний конструктор базового класу, то компілятор автоматично викличе конструктор базового класу за замовчуванням.

Наявність private-членів базового класу, доступних лише методам базового класу, який завжди зручно. Повинно існувати багато випадків, коли може знадобитися мати приватні члени базового класу, які можуть бути доступні у похідному класі. І, логічно очікувати, що C++ надає таку можливість.

**Оголошення членів класу як protected**

Крім специфікаторів доступу *public* та *private*, члени класу можна також оголошувати як *protected* (захищені). Всередині класу ключове слово *protected* забезпечує той самий ефект, як і слово *private*: члени класу з цією специфікацією можуть бути доступні лише методам цього класу та його дружнім функціям.

Якщо елементу встановлено специфікатор доступу *protected*:

-у успадкованих класах до нього відкритий прямий доступ

- його не можна викликати через об'єкт: person1.m_weight

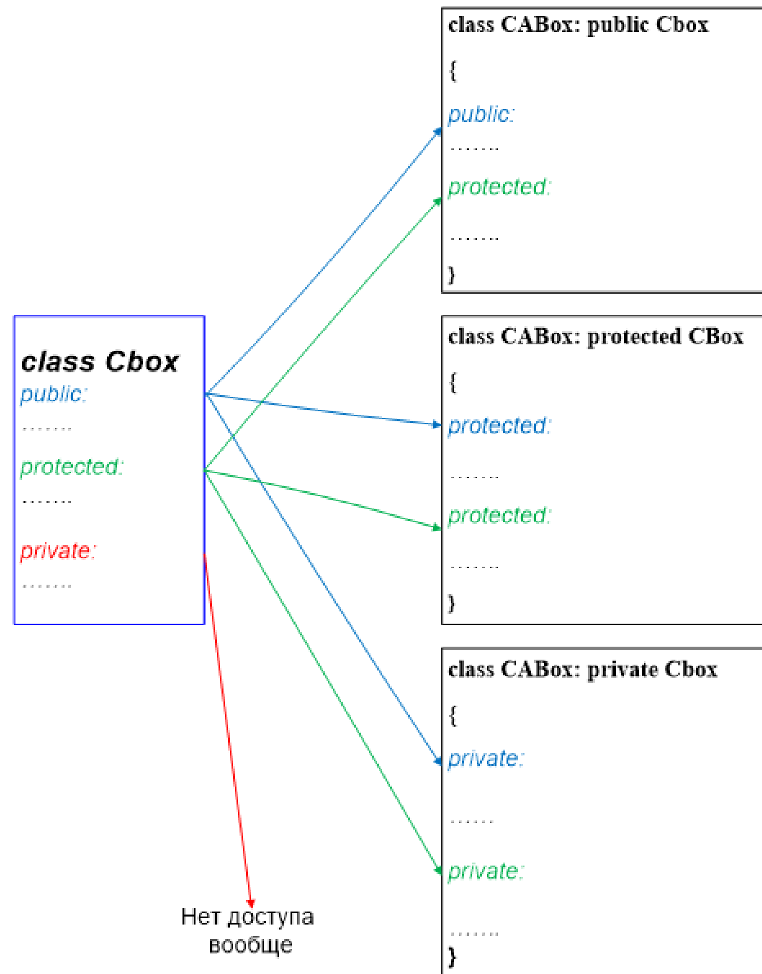
**Приклад_10.7.**

//Розглянемо можливість доступу до елементів із різними специфікаторами доступу (public, private, protected):

```
#include <iostream>
#include <string>
using namespacestd;
```

```
class Human
{
public:
	string m_Name;
private:
	int m_age;	// Доступно тільки для класу Human
protected:
	int m_weight;
 public:
Human(std::string Name="",int age=0,int weight=0)
: m_Name(Name), m_age(age),m_weight(weight)
{
	std::cout<< "Constr Human" <<std::endl;
}
~Human()
{}
};
class Student:public Human
{
 String m_Group;
public:
Student(std::string Name="",int age= 0,int weight= 0, std::string Group="")
:Human(Name,age,weight)
{
	m_Group= Group;
	std::cout<< "Constr Student" <<std::endl;
}
 моid print()
{
std::cout<<m_Name<<std::endl;	//специфікатор доступу
std::cout<<m_Group<<std::endl;
std::cout<<m_weight<<std::endl;	//специфікатор доступу protected
}
};
class Professor:public Human
{
	string m_Subject;
};
int main()
{
	setlocale(LC_ALL,"ru");

	Student person1("Вася",19,100,"ТА2113");
	person1.print();
	std::cout<<std::endl;
	std::cout<< "специфікатор доступу public" <<std::endl;

// Специфікатор доступу public:
	std::cout<<person1.m_Name<<std::endl;

 //std::cout << "специфікатор доступу protected" << std::endl;
// Немає доступу через об'єкт:
```

```
// std::cout << person1.m_weight << std::endl;
return0;
}
```




## 11. ВІРТУАЛЬНА ФУНКЦІЯ. ПОЛІФОРМІЗМ.

Розглянемо поведінку успадкованих методів та його відносин з методами похідного класу.

Створимо базовий клас коробок CBox, який включає функцію визначення об'єму *Volume()*

Додамо функцію *Show()* до класу CBox, щоб вивести об'єкт CBox. Спрощена версія класу стане такою:

```
class CBox
{
protected:
 doublem_l;
 doublem_w;
 doublem_h;
public:
CBox(double l= 1.0double w= 1.0double h= 1.0): m_l(l), m_w(w), m_h(h)
{
cout<< "Constr1" <<endl;
```

```
}
~CBox()
{
cout<< "Destr1" <<endl;
}
 doubleVolume()
{
 returnm_l*m_w*m_h;
}
 voidShow()
{
cout<< "V:" <<Volume()<<endl;
}
};
```

Тепер ви можете вивести корисний обсяг об'єкта CBox, просто викликавши функцію Show() з будь-яким об'єктом, який вам потрібен. Конструктор встановлює значення полів у списку ініціалізації, тому ніякі оператори в тілі функції не потрібні. Поля залишаються колишніми та специфікуються як protected, тому вони доступні методам будь-якого похідного класу.

Припустимо, що ви хочете створити похідний клас для моделювання ящиків іншого виду під назвою CGlassBox, щоб зберігати скляний посуд.

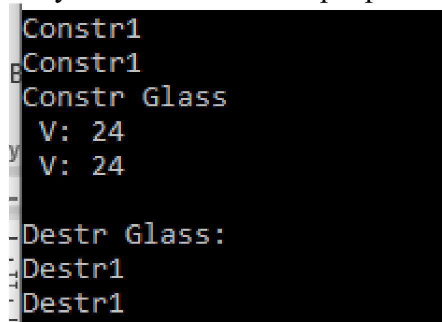
Вміст крихкий, і оскільки для його запобігання додається пакувальний матеріал, ємність такого ящика буде меншою, ніж базового об'єкта CBox. Тому вам знадобиться інша функція Volume (), щоб врахувати цю обставину, і ви додаєте її до похідного класу:

**Приклад_11.1.**

```
#include <iostream>
using namespace std;
class CBox
{
protected:
 double m_l;
 double m_w;
 double m_h;
public:
CBox(double l= 1.0double w= 1.0double h= 1.0): m_l(l), m_w(w), m_h(h)
{
cout<< "Constr1" <<endl;
}
~CBox()
{
cout<< "Destr1" <<endl;
}

 double Volume()
{
	return m_l*m_w*m_h;
}

 voidShow()
```

```
{
        cout<< "V:" <<Volume()<<endl;
}
};

class CGlass:public CBox
{
public:
 double Volume()
{
        return0.85*m_l*m_w*m_h;
}
CGlass(double l,double w,double h) :CBox(l,w,h)
{
        cout<< "Constr Glass" <<endl;
}
~CGlass()
{
cout<< "Destr Glass:" <<endl;
}
};

int main()
{
 CBox box1(2.0, 3.0, 4.0);
 CGlass box2(2.0, 3.0, 4.0);
box1.Show();
 // виклик Volume ( ) методом Show () встановлений компілятором за версією,
 // визначеної в базовому класі.
box2.Show();
cout<<endl;
 return 0;
}
```

Результат виконання програми:


```
Constr1
Constr1
Constr Glass
 V: 24
 V: 24

Destr Glass:
Destr1
Destr1
```

Ймовірно, у похідному класі будуть інші додаткові члени, але для простоти ми не будемо їх додавати, а зосередимося на тому, як працюють успадковані функції. Конструктор для об'єктів похідного класу просто викликає конструктор базового класу у списку ініціалізації, щоб встановити значення полів. Жодних додаткових операторів у його тілі не потрібно. Ви включили нову версію функції Volume () замість версії з базового класу. Ідея полягає в тому, щоб змусити успадковану функцію Show() звертатися до версії функції

обчислення об'єму Volume() похідного класу, коли ви запускаєте її з об'єктом похідного класу CGlassBox.

Причина некоректного висновку в тому, що виклик Volume( ) у методі ShowVolume() встановлений компілятор раз і назавжди за версією, визначеною в базовому класі. Show() - метод базового класу, і коли компілюється клас CBox, то виклик Volume() у ньому дозволяється у момент компіляції як виклик методу Volume() базового класу; компілятор не має уявлення про жодну іншу функцію Volume (). Це називається статичним дозволом виклику функції, оскільки виклик фіксований до виконання програми. Іноді це називають раннім зв'язуванням, оскільки певна обрана функція прив'язується до виклику з функції Show () під час компіляції програми.

**Виправлення CGlassBox**

**Віртуальна функція**- це функція в базовому класі, оголошена з допомогою ключового слова virtual. Якщо ви визначите функцію базового класу як virtual, і у похідному класі є інше визначення цієї функції, це повідомить компілятор, що вам не потрібне статичне компонування цієї функції. Що вам потрібно - так це щоб вибір функції, яка повинна бути викликана в будь-якій заданій точці програми, був заснований на тип об'єкта, для якого вона викликається.

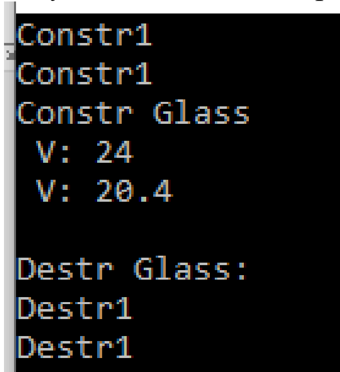
Щоб цей приклад виправдав надії, потрібно просто додати ключове слово virtual до визначення функції Volume () у цих двох класах

**Приклад_11.2.**

```
//Використання віртуальної функції

#include <iostream>
using namespace std;
class CBox
{
protected:
 double m_l;
 double m_w;
 double m_h;
public:
	//Конструктор
CBox(double l= 1.0, double w= 1.0, double h= 1.0): m_l(l), m_w(w), m_h(h)
{
	cout<< "Constr1" <<endl;
}
~CBox()
{
	cout<< "Destr1" <<endl;
}

 virtual double Volume() //Функція обчислення обсягу об'єкта CBox
{
	 return m_l*m_w*m_h;
}

 void Show()//Функція для відображення обсягу об'єкта
{
	cout<< "V:" <<Volume()<<endl;
}
```

```
};
class CGlass:public CBox     //Виробний клас
{
public:

//функція похідного класу для обчислення обсягу CGlassBox
//резервуюча 15% на упаковку
 virtual double Volume()
{
 return 0.85*m_l*m_w*m_h;
}
//Конструктор
CGlass(double l,double w,double h) :CBox(l,w,h)
{
        cout<< "Constr Glass" <<endl;
}
~CGlass()
{
cout<< "Destr Glass:" <<endl;
}
};
Int main()
{
 CBox box1(2.0, 3.0, 4.0);     //Оголошення базової скриньки
 CGlass box2(2.0, 3.0, 4.0);  //Оголошення похідної скриньки

box1.Show();          //Відобразити обсяг базової скриньки

box2.Show();          //Відобразити обсяг похідної скриньки
cout<<endl;
 return 0;
}
```

Результат виконання програми:


```
Constr1
Constr1
Constr Glass
 V: 24
 V: 20.4

Destr Glass:
Destr1
Destr1
```

**Опис отриманих результатів.**

Тепер програма робить те, що ви хотіли. Перший виклик функції *Show()* з об'єктом *myBox* типу *CBox* звертається до версії *Volume()* із класу *CBox*. Другий виклик із об'єктом my *GlassBox* типу *CGlassBox* активізує версію, визначену у похідному класі.

Зверніть увагу, що, незважаючи на те, що ви помістили ключове слово *virtual* на визначення похідного класу функції Volume (), робити це не обов'язково.   Визначення

базової версії функції як *virtual* цілком достатньо. Однак все ж таки бажано, щоб ви вказували це ключове слово для віртуальних функцій у похідних класах, оскільки це зробить ясним будь-кому, хто читатиме визначення похідного класу, що йдеться про віртуальні функції, які при виконанні вибираються динамічно.

Щоб функція поводилася як віртуальна, вона повинна мати те саме ім'я, список параметрів і тип повернення у всіх похідних класах, як у базовому класі, і якщо у базовому класі функція оголошена як*const*, функція похідного класу також має бути const. Якщо ви спробуєте використовувати інші типи параметрів або повернення або оголосити функцію в одному місці const, а в іншому - ні, то в цьому випадку механізм віртуальних функцій не працюватиме. Функція буде компонована статично та фіксована під час компіляції.

Робота віртуальних функцій – виключно потужний механізм. Термін **поліморфізму** зв'язку з об'єктно-орієнтованим програмуванням має відношення до можливостей віртуальних функцій.

**Поліморфізм** (polymorphism) (від грецького polymorphos) - це властивість, що дозволяє одне й те саме ім'я використовуватиме вирішення двох чи більше схожих, але технічно різних завдань. Метою поліморфізму стосовно об'єктно-орієнтованого програмування є використання одного імені для завдання загальних для класу дій. Виконання кожної конкретної дії визначатиметься типом даних.

Виклик віртуальних функцій призводить до різного ефекту залежно від виду об'єкта, якого вона викликана.

Слід зазначити, що функція *Volum()* у похідному класі CGlassBox фактично приховує версію цієї функції з базового класу від функцій похідного класу. Якщо ви хочете викликати версію *Volume()* базового класу з функції похідного класу, то повинні використовувати операцію дозволу контексту для звернення до функції, наприклад, CBox:: Volume ().

Наприклад:

cout << " V box2 (CBox )=" << **box2.CBox::Volume()** << endl;