МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Н. Ю. Любченко, М. О. Соболь,
А. О. Подорожняк, Р. В. Пугачов, О. В. Любченко

ОСНОВИ ООП C++ В ПРИКЛАДАХ

Частина 2

Навчально-методичний посібник
для студентів комп'ютерних спеціальностей
вищих навчальних закладів

Затверджено редакційно-видавничою
радою НТУ «ХПІ»,
протокол № 1 від 19.02.2026 р.

Харків
НТУ «ХПІ»
2026

УДК 004.43
Л 75

Рецензенти:
А. А. Коваленко, д-р техн. наук, проф.,
Харківський національний університет радіоелектроніки;
Д. Б. Єльчанінов, канд. техн. наук, доцент,
Національний технічний університет «Харківський політехнічний інститут»

Любченко Н. Ю.

Л 75 Основи ООП С++ в прикладах. Частина 2: навчально-методичний посібник для студентів комп'ютерних спеціальностей вищих навчальних закладів / Н. Ю. Любченко, М. О. Соболь, А. О. Подорожняк, Р. В. Пугачов, О. В. Любченко. Харків : НТУ «ХПІ», 2026. 219 с.

ISBN 978-617-05-0623-8

У навчально-методичному посібнику розглянуті більш просунуті аспекти мови програмування С++, такі, як шаблонні функції та класи, стандартна бібліотека шаблонів (STL), алгоритми, лямбда-вирази. Посібник містить теоретичні матеріали, приклади програм у вигляді добре документованого початкового програмного коду.

Призначено для студентів, викладачів комп'ютерних спеціальностей та всіх, хто прагне поглибити знання з програмування.

Табл. 3. Бібліогр. 15 назв.

УДК 004.43

ISBN 978-617-05-0623-8

ВСТУП

Навчально-методичний посібник з курсу «Об'єктно-орієнтоване програмування» призначений для студентів вищих навчальних закладів, які опановують основи програмування в контексті об'єктно-орієнтованої парадигми.

Друга частина посібника присвячена поглибленому вивченню складних концепцій та інструментів мови C++, які дозволяють розширити функціональні можливості програмного забезпечення. У цій частині розглядаються такі важливі теми, як шаблонні функції та класи, що забезпечують можливість написання узагальненого коду, а також бібліотека стандартних шаблонів (STL), яка надає потужні засоби для роботи з даними.

Особлива увага приділяється функторам, алгоритмам STL, лямбда-виразам та іншим сучасним засобам, що дозволяють створювати більш компактний та продуктивний код. Також студенти ознайомляться з важливими аспектами тестування програм та використанням класу STRING для роботи з рядками.

Завдання та матеріали цієї частини допоможуть студентам не лише освоїти просунуті можливості C++, але й навчать їх застосовувати у реальних проєктах, підвищуючи продуктивність програмування та якість кінцевих рішень.

Посібник стане корисним не лише студентам, а й викладачам та програмістам-початківцям, які прагнуть поглибити свої знання в об'єктно-орієнтованому підході до програмування.

8. ШАБЛОННІ ФУНКЦІЇ ТА КЛАСИ

Мета: Оволодіння практичними навичками роботи з шаблонними функціями та класами в мові С++

Теми для попереднього опрацьовування:

- Перевантаження функцій.
- Шаблонні функції та класи.

Теоретичні відомості

Шаблон є функцією (*template function*) або класом (*template class*), реалізованими для одного або декількох типів даних, які не відомі у момент написання коду. При використанні шаблону, як аргументи йому явно або неявно передаються конкретні типи даних.

Узагальнена функція (***шаблон функції*** – *template function*) визначає універсальну сукупність операцій, застосовних до різних типів даних. Тип даних, з якими працює функція, передається як параметр. Багато алгоритмів носять універсальний характер і не залежать від типу даних, якими вони оперують.

Визначення шаблонної функції виглядає таким чином :

```
template  <class  Ттип>
тип_значення <ім'я_функції> (<список_параметрів>)
{
     // Тіло функції
}
```

Тут параметр `Ттип` – ім'я типу – задає тип даних, з яким працює функція. Цей параметр можна використовувати і усередині функції, проте при створенні конкретної версії узагальненої функції компілятор автоматично підставить замість нього фактичний тип.

Традиційно узагальнений тип задається за допомогою ключового слова *class*, хоча замість нього можна застосовувати ключове слово *typename*.

***Приклад* 1.** Шаблони функцій. Узагальнена функція, що міняє місцями дві змінні.

```
template <class X>             // Шаблон функції
  void swapargs(X &a, X &b)
{
    X temp;
    temp = a;   a = b;   b = temp;
}

int main()
{
   int    i=10,   j=20;
   double x=10.1, y=23.3;
   char   a='q',  b='w';
   cout << "Початкові значення i, j: " << i <<' '<< j ;
   cout << "Початкові значення x, y: " << x <<' '<< y ;
   cout << "Початкові значення a, b: " << a <<' '<< b ;

   swapargs (i, j);
   swapargs (x, y);
   swapargs (a, b);

   cout << "Переставлені значення i,j:"<<i<<' ' << j;
   cout << "Переставлені значення x,y:"<<x<<' ' << y;
   cout << "Переставлені значення a,b:"<<a<<' ' << b;
}
```

Функція з двома узагальненими типами

Використовуючи список, елементи якого розділені комами, можна визначити декілька узагальнених типів даних в операторі *template*.

Наприклад, в наступній програмі створюється шаблонна функція, що має *два* узагальнених типа.

```
template <class type1, class type2>
 void myfunc(type1  x,  type2  y)
{
      cout  <<  x << "  " <<  y  <<  '\n';
}
```

```
int main()
{
   myfunc(10, "Hello");
   myfunc(98.6,   19L);
}
```

Шаблонна функція може бути *перевантажена іншим шаблоном* або нешаблонною функцією з таким же ім'ям, але з іншим набором параметрів.

```
// Шаблон для порівняння
template   <class A>
A _min_ (const A &x, const A &y)
{
    return x < y ? x : y;
}

// Нешаблонна функція для порівняння рядків
 int _min_ (const char* x, const char* y)
{
    return  strcmp(x, y);
}

int main ( )
{
    cout << _min_(3, 4) << endl;
    cout << _min_(3.2, 0.4) << endl;
    cout << _min_("66", "55555") << endl;     // 1
    cout << _min_("55555", "66") << endl;     //-1
    cout << _min_("51666", "55555") << endl;  //-1
}
```

Шаблони класів. Узагальнені класи корисні, якщо логіка класу не залежить від типу даних. Оголошення узагальненого класу має наступний вигляд :

```
template  <class Ттип>
  class  <ім'я_класу>
{
  ......
};
```

Тут параметр Ттип задає тип даних, який уточнюється при створенні екземпляра класу. При необхідності можна визначити декілька узагальнених типів, використовуючи список імен, розділених комою.

Конкретний екземпляр узагальненого класу створюється за допомогою наступної синтаксичної конструкції :

```
< ім'я_класу >  <тип>  <ім'я_об'єкту>;
```

Тут параметр <тип> задає тип даних, якими оперує клас. Функції-члени узагальненого класу автоматично стають узагальненими. Для їх оголошення не обов'язково використовувати ключове слово *template*.

***Приклад* 2.** Шаблони класів.

```
template   <class T>
class A
{
      T x;
  public:
      A() :x(0) {}
      A(T a) :x(a) {}
      void show() {
          cout << x << endl;
      }
};
int main ( )
{
    A <int> ob;
    ob.show();

    A <char> ob1(5);
    ob1.show();
}
```

***Приклад* 3.** Шаблони класів. Використання двох узагальнених типів даних.

```
template <class Type1, class Type2>
class myclass
{
    Type1 i;
```

```
    Type2 j;
  public:
    myclass(Type1  a, Type2  b)
    {
         i = a;  j = b;
    }
    void show(){
         cout << i << ' ' << j << '\n';
   }
};

int main()
{
  myclass <int, double> ob1(10, 0.23);
  myclass <char, string> ob2('X',"Шаблоны");
  ob1.show();           // Виводимо ціле і дійсне число
  ob2.show();           // Виводимо символ і рядок
}
```

***Приклад* 4.** Використання класу виключення і шаблонних функцій. Створити клас, що містить масив з елементів різних типів. Перевантажити оператор [ ] для перевірки виходу індексу масиву за межі діапазону. Забезпечити можливість ініціалізації і друку масиву у функції *main*() через перевантаження [ ].

Використовувати механізм обробки виняткових ситуацій і роздільну компіляцію.

```
// Header.h
#include <string>
using namespace std;
#define n 6

class myException
{
     string errMsg;
  public:
    myException(string s);
    string what();
};
```

```
template <class T>
class AR
{
    T *mas;
  public:
    AR();
    ~AR();
    T & operator[](int i);
};

//Source.cpp
#include "Header.h"
#include <iostream>
using namespace std;

myException::myException(string s) {
    errMsg = s;
}

string myException::what(){
    return errMsg;
}

template AR <int>;             // явне оголошення шаблонів
template AR <float>;

         // Або треба оголошувати для кожної функції для
         // певного типу
         // template AR<int>::AR();
         // template AR<int>::~AR();
         // template int& AR<int>::operator[](int i);
         // template AR<float>::AR();
         // template AR<float>::~AR();
         // template int& AR<float>::operator[](int i);

template <class T> AR<T>::AR()
{
   mas = new T[n];
  for (int i = 0; i < n; i++) {
```

```
        mas[i] = (T)(i*1.1);
        //  a6o  mas[i] = static_cast<T>(i * 1.1);
  }
}

template <class T> AR<T>::~AR()
{
  delete[]mas;
}

template <class T>
T & AR<T>::operator[](int i)
{
    if (i < 0 || i >= n)
        throw myException("Error []");
    else
        return mas[i];
}

//main.cpp
#include "Header.h"
#include <iostream>
using namespace std;

int main()
{
  try
 {
        AR <int> ob;
        for (int i = 0; i < n; i++)         //n = 6
            cout << ob[i] << " ";

        cout << endl;

        for (int i = 0; i < n+2; i++){
            ob[i] = i * 10;
            cout << ob[i] << " ";
        }
  }
```

```
catch (myException e) {
    cout << e.what()<<endl;
}

cout << endl;
```

```
0  1  2  3  4  5
0  10  20  30  40  50  Error []
```

```
try
 {
        AR <float> ob1;
        for (int i = 0; i < n; i++)
            cout << ob1[i] << " ";
        cout << endl;
        for (int i = -100; i < n; i++) {
            ob1[i] = i * 10;
            cout << ob1[i] << " ";
        }
}
 catch (myException e) {
       cout << e.what() << endl;
 }
} //main
```

```
0  1.1  2.2  3.3  4.4  5.5
Error []
```

Задачі

Загальні умови

Кожне завдання містить приклад використання шаблонних функцій та класів в мові С++, і може містити умови вирішення. Після формулювання завдання надано рішення у вигляді програмного коду.

1. Створити шаблонний клас *PairOfNumbers*. У класі наявні два поля(числа).

Методи класу:

– конструктор з параметрами;

– сетери для полів класу;

– функція пошуку максимального числа;

– функція *swap*;

– функція друку чисел.

Методи класів реалізувати у *cpp* файлі, перед цим явно оголосити два шаблони з типами даних:

– *int*;

– *double*.

Розв'язання задачі 1:

PairOfNumbers.h

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

template<typename Type>
class PairOfNumbers
{
private:

	//Перше число
	Type firstNumber;

	//Друге число
	Type secondNumber;

public:

	//Конструктор з параметрами
	PairOfNumbers(const Type& firstNumber,
		const Type& secondNumber);

	//Сетери для полів класу
	void setFirstNumber(const Type& number);
	void setSecondNumber(const Type& number);

	//Функція пошуку максимального числа
	Type getMax() const;

	//Функція зміни чисел
	void swapNumbers();

	//Функція друку чисел
	void printNumbers() const;
```

```
};
```

PairOfNumbers.cpp

```
#include "PairOfNumbers.h"

//Явне оголошення шаблонів
template PairOfNumbers<int>;
template PairOfNumbers<double>;

//Конструктор з параметрами
template<typename Type>
PairOfNumbers<Type>::PairOfNumbers(const Type& firstNumber,
                                   const Type& secondNumber)
{
	this->firstNumber = firstNumber;
	this->secondNumber = secondNumber;
}

//Сетери для полів класу
template<typename Type>
void PairOfNumbers<Type>::setFirstNumber(const Type& number)
{
	this->firstNumber = number;
}

template<typename Type>
void PairOfNumbers<Type>::setSecondNumber(const Type& number)
{
	this->secondNumber = number;
}

//Функція пошуку максимального числа
template<typename Type>
Type PairOfNumbers<Type>::getMax() const
{
	return firstNumber > secondNumber ? firstNumber
	                                  : secondNumber;
}

//Функція зміни чисел
template<typename Type>
void PairOfNumbers<Type>::swapNumbers()
{
	Type temp = firstNumber;
```

```
        firstNumber = secondNumber;
        secondNumber = temp;
}

//Функція друку чисел
template<typename Type>
void PairOfNumbers<Type>::printNumbers() const
{
        cout << "Перше число: " << firstNumber
                << " ,друге число: " << secondNumber << endl;
}
```

Practice_task_1.cpp

```
#include <Windows.h>
#include "PairOfNumbers.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    PairOfNumbers<int> numbersInt(5, 11);
    PairOfNumbers<double> numbersDouble(1.89, 99.17);

    cout << " --- Друк чисел ---\n";
    numbersInt.printNumbers();
    numbersDouble.printNumbers();

    cout << "\n--- Максимальне число ---\n";
    cout << "Максимальне  число(int): " << numbersInt.getMax()
                                        << "\n";
    cout << "Максимальне  число(double): "
         << numbersDouble.getMax() << "\n";

    cout << "\n--- Зміна чисел ---\n";
    numbersInt.swapNumbers();
    numbersDouble.swapNumbers();

    //Друк чисел
    numbersInt.printNumbers();
    numbersDouble.printNumbers();
}
```

2. Створити клас *Exception*, за допомогою якого будуть оброблятися винятки. Параметром класу є повідомлення про виключення. Методом класу є гетер для поля класу.

Створити шаблонний клас *Array*. Поля класу:

– покажчик на масив;

– розмір масиву.

Методи класу *Array*:

– конструктор з параметром(розмір масиву);

– деструктор;

– гетер для розміру масиву;

– функція *push*, яка додає новий елемент у кінець масиву;

– перевантаження оператору квадратних дужок.

Реалізовувати методи шаблонного класу *Array* у файлі *cpp*. Явно оголосити два шаблони з типами даних:

– *int*;

– *float*.

Розв'язання задачі 2:

Exception.h

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

class Exception
{
private:

	//Повідомлення
	string message;

public:

	//Конструктор з параметром
	Exception(const string& message);

	//Функція, що повертає повідомлення
	string what() const;
};
```

Exception.cpp

```
#include "Exception.h"

//Конструктор з параметром
Exception::Exception(const string& message)
{
	this->message = message;
}

//Функція, що повертає повідомлення
string Exception::what() const
{
	return this->message;
}
```

Array.h

```
#include "Exception.h"

template<typename Type>
class Array
{
private:

	//Покажчик на масив
	Type* array;

	//Розмір масиву
	int arraySize;

public:

	//Конструктор з параметром
	Array(const int arraySize);

	//Деструктор
	~Array();

	//Гетер для розміру масиву
	int getSize() const;

	//Перевантаження оператора[]
	Type& operator[](const int index);
```

```
	//Функція додавання числа
	void push(const Type number);
};
```

Array.cpp

```
#include "Array.h"

//Явне оголошення шаблонів
template Array<int>;
template Array<float>;

//Конструктор з параметром
template<typename Type>
Array<Type>::Array(const int arraySize)
{
	if (arraySize < 1)
		throw Exception("Некоректний розмір масиву!");

	this->arraySize = arraySize;
	this->array = new Type[arraySize];

	for (int i = 0; i < arraySize; i++)
		*(array + i) = (Type)((i + 1) * 1.1);
}

//Деструктор
template<typename Type>
Array<Type>::~Array()
{
	delete[] this->array;
}

//Гетер для розміру масиву
template<typename Type>
int Array<Type>::getSize() const
{
	return this->arraySize;
}

//Перевантаження оператора[]
template<typename Type>
Type& Array<Type>::operator[](const int index)
{
	if (index < 0 && index >= arraySize)
```

```
                throw Exception("Некоректний індекс!");

        return *(array + index);
}

//Функція додавання числа
template<typename Type>
void Array<Type>::push(const Type number)
{
        Type* tempArray = new Type[arraySize + 1];

        for (int i = 0; i < arraySize; i++)
                *(tempArray + i) = *(array + i);

        *(tempArray + arraySize) = number;

        delete[] this->array;
        this->array = tempArray;
        this->arraySize++;
}
```

Practice_task_2.cpp

```
#include <Windows.h>
#include "Array.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    srand(time(NULL));

        const int arraySize = 5;

        try
        {

                //Створення об'єкту шаблонного класу з типом int
                Array<int> arrayInt(arraySize);

                cout << " --- Масив типу int ---\n";
                for (int i = 0; i < arrayInt.getSize(); i++)
                {
                        cout << arrayInt[i] << " ";
                }
```

```
            //Створення об'єкту шаблонного класу з типом
            //float
            Array<float> arrayFloat(arraySize);

            cout << "\n\n--- Масив типу float ---\n";
            for (int i = 0; i < arrayFloat.getSize(); i++)
            {
                  cout << arrayFloat[i] << " ";
            }

            //Додавання нового елемента у кінець arrayFloat
            arrayFloat.push(99.99);

            cout << "\n\n---Змінений масив типу float---\n";
            for (int i = 0; i < arrayFloat.getSize(); i++)
            {
                  cout << arrayFloat[i] << " ";
            }
      }
      catch (const Exception& exception)
      {
            cout << exception.what() << endl;
      }
}
```

3. Створити шаблонний клас *Circle*. Полями класу *Circle* є:

– координата x;

– координата y;

– радіус.

Методи класу:

– конструктор з параметрами(координати x, y та радіус);

– перевантаження конструктора з параметрами(радіус);

– конструктор за замовчуванням;

– функція підрахунку довжини окружності;

– функція підрахунку площі;

– група "гетерів" для полів класу.

Створити шаблонний клас *Sphere*, який успадковується від шаблонного класу *Circle*. Полем класу є додаткова координата. Методи класу *Sphere*:

– конструктор з параметрами;

– перевантаження конструктора з параметрами(радіус);

– функція підрахунку площі поверхні;

– функція підрахунку об'єму;

– гетер для поля класу.

Розв'язання задачі 3:

Circle.h

```
#define PI 3.1415

template <typename Type>
class Circle
{
protected:

	//Координати
	Type x;
	Type y;

	//Радіус
	Type radius;

public:

	//Конструктор з параметрами
	Circle(const Type x, const Type y,
		const Type radius);

	//Перевантаження конструктора з параметрами
	Circle(const Type radius);

	//Конструктор за замовчуванням
	Circle();

	//Фунція підрахунку довжини окружності
	Type getCircumference() const;

	//Функція підрахунку площі
```

```
        Type getArea() const;

        //Група "гетерів" для полів класу
        Type getX() const;
        Type getY() const;
        Type getRarius() const;
};
```

Circle.cpp

```
#include "Circle.h"

//Явне оголошення шаблонів
template Circle<int>;
template Circle<double>;

//Конструктор з параметрами
template<typename Type>
Circle<Type>::Circle(const Type x, const Type y,
                     const Type radius)
{
        this->x = x;
        this->y = y;
        this->radius = radius;
}

//Перевантаження конструктора з параметрами
template<typename Type>
Circle<Type>::Circle(const Type radius)
{
        this->x = 0;
        this->y = 0;
        this->radius = radius;
}

//Конструктор за замовчуванням
template<typename Type>
Circle<Type>::Circle()
{
        this->x = 0;
        this->y = 0;
        this->radius = 1;
}

//Фунція підрахунку довжини окружності
```

```
template<typename Type>
Type Circle<Type>::getCircumference() const
{
	return 2 * PI * this->radius;
}

//Функція підрахунку площі
template<typename Type>
Type Circle<Type>::getArea() const
{
	return PI * radius * radius;
}

//Група "гетерів" для полів класу
template<typename Type>
Type Circle<Type>::getX() const
{
	return this->x;
}

template<typename Type>
Type Circle<Type>::getY() const
{
	return this->y;
}

template<typename Type>
Type Circle<Type>::getRarius() const
{
	return this->radius;
}
```

Sphere.h

```
#include "Circle.h"

template<typename Type>
class Sphere : public Circle<Type>
{
protected:

	//Додаткова координата
	Type z;

public:
```

```
        //Конструктор з параметрами
        Sphere(const Type x, const Type y, const Type z,
               const Type radius);

        //Перевантаження конструктора з параметрами
        //Використаний параметр за замовчуванням
        Sphere(const Type radius = 1);

        //Функція підрахунку площі поверхні
        Type getSurfaceArea() const;

        //Функція підрахунку об'єму
        Type getVolume() const;

        //Гетер поля класу
        Type getZ() const;
};
```

Sphere.cpp

```
#include "Sphere.h"

//Явне оголошення шаблонів
template Sphere<int>;
template Sphere<double>;

//Конструктор з параметрами
template<typename Type>
Sphere<Type>::Sphere(const Type x, const Type y,
                     const Type z, const Type radius):
        //Явний виклик конструктору батьківського класу
        Circle<Type>::Circle(x, y, radius)
{
        this->z = z;
}

//Перевантаження конструктора з параметрами
template<typename Type>
Sphere<Type>::Sphere(const Type radius)
{
        //При виклику функцій батьківського
        //шаблонного класу або зверненні до поля
        //обов'язково використовувати this.
```

```
        this->x = 0;
        this->y = 0;
        this->z = 0;
        this->radius = radius;
}

//Функція підрахунку площі поверхні
template<typename Type>
inline Type Sphere<Type>::getSurfaceArea() const
{
        return 4 * PI * this->radius * this->radius;
}

//Функція підрахунку об'єму
template<typename Type>
inline Type Sphere<Type>::getVolume() const
{
        return 4 / 3 * PI * this->radius * this->radius;
}

//Гетер поля класу
template<typename Type>
inline Type Sphere<Type>::getZ() const
{
        return this->z;
}
```

Practice_task_3.cpp

```
#include <Windows.h>
#include <iostream>
#include "Sphere.h"

using namespace std;

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Circle<int> circle(10);

    cout << " --- Інформація про коло(int) ---\n";
    cout << "Координата x: " << circle.getX() << endl;
    cout << "Координата y: " << circle.getY() << endl;
```

```
    cout << "Радіус: " << circle.getRarius() << endl;
    cout << "Довжина окружності: " << circle.getCircumference()
         << endl;
    cout << "Площа: " << circle.getArea() << endl;

    Sphere<double> sphere(3.5, 4.2, -1, 5.8);

    cout << "\n--- Інформація про сферу(double) ---\n";
    cout << "Координата x: " << sphere.getX() << endl;
    cout << "Координата y: " << sphere.getY() << endl;
    cout << "Координата z: " << sphere.getZ() << endl;
    cout << "Радіус: " << sphere.getRarius() << endl;
    cout << "Площа поверхні: " << sphere.getSurfaceArea()
         << endl;
    cout << "Об'єм: " << sphere.getVolume() << endl;
}
```

4. Створити шаблонний клас *SmartPointer*, який реалізовує роботу покажчика та здійснює автоматичний контроль за пам'яттю. Полем класу є покажчик на об’єкт. Методи класу:

– конструктор з параметром;

– конструктор копіювання, за допомогою якого передається управління об'єктом;

– деструктор;

– перевантаження оператора(=);

– перевантаження оператора(*);

– перевантаження оператора(->);

– функція перевірки на наявність об'єкту.

Реалізовувати клас *SmartPointer* у заголовному файлі. Роботу з покажчиком реалізувати за допомогою семантики переміщення, тобто володіння об'єктом передається від одного покажчика іншому.

Створити клас *Account*. Поля класу *Account*:

– пошта;

– пароль.

Методи класу:

– конструктор з параметрами;

– функція зміни паролю;

– функція друку інформації.

Для демонстрації роботи динамічно виділити пам'ять для об'єкту класу *Account*. Створити пару покажчиків та передати управління з одного покажчика іншому.

Розв'язання задачі 4:

SmartPointer.h

```
template<typename Type>
class SmartPointer
{
private:

	//Покажчик на об'єкт
	Type* ptr;

public:

	//Конструктор з параметром
	SmartPointer(Type* ptr = nullptr);

	//Конструктор копіювання
	SmartPointer(SmartPointer& ptr);

	//Деструктор
	~SmartPointer();

	//Перевантаження оператора(=)
	SmartPointer<Type>& operator=(SmartPointer& object);

	//Перевантаження оператора(*)
	Type& operator*() const;

	//Перевантаження оператора(->)
	Type* operator->() const;

	//Функція перевірки на порожнечу
	bool isNull() const;
};

//Конструктор з параметром
template<typename Type>
SmartPointer<Type>::SmartPointer(Type* ptr)
```

```
{
	this->ptr = ptr;
}

//Конструктор копіювання
template<typename Type>
SmartPointer<Type>::SmartPointer(SmartPointer& object)
{
	//Передача управління поточному покажчику
	this->ptr = object.ptr;

	//Розриваємо зв'язок об'єкта з попереднім покажчиком
	object.ptr = nullptr;
}

//Деструктор
template<typename Type>
SmartPointer<Type>::~SmartPointer()
{
	if(!isNull)
		delete this->ptr;
}

//Перевантаження оператора(=)
template<typename Type>
SmartPointer<Type>& SmartPointer<Type>::operator=
                                        (SmartPointer& object)
{
	//Якщо самоприсвоєння
	if (&object == this)
		return *this;

	//Видаляємо дані про минулий об'єкт
	if (!isNull)
		delete this->ptr;

	//Передача управління об'єктом
	this->ptr = object.ptr;
	object.ptr = nullptr;

	//Якщо ланцюг привласнень
	return *this;
}

//Перевантаження оператора(*)
```

```
template<typename Type>
Type& SmartPointer<Type>::operator*() const
{
	return *ptr;
}

//Перевантаження оператора(->)
template<typename Type>
inline Type* SmartPointer<Type>::operator->() const
{
	return ptr;
}

//Функція перевірки на наявність об'єкту
template<typename Type>
inline bool SmartPointer<Type>::isNull() const
{
	return ptr == nullptr;
}
```

Account.h

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

class Account
{
private:

	//Пошта
	string mail;

	//Пароль
	string password;

public:

	//Конструктор з параметром
	Account(const string& mail,
		const string& password);

	//Функція зміни паролю
	void setPassword(const string& newPassword);

	//Функція друку інформації
```

```
        void printInformation() const;
};
```

Account.cpp

```
#include "Account.h"

//Конструктор з параметром
Account::Account(const string& mail, const string& password)
{
        this->mail = mail;
        this->password = password;
}

//Функція зміни паролю
void Account::setPassword(const string& newPassword)
{
        this->password = newPassword;
}

//Функція друку інформації
void Account::printInformation() const
{
        cout << "Пошта: " << mail
                << "\nПароль: " << password << endl;
}
```

Practice_task_4.cpp

```
#include <Windows.h>
#include "SmartPointer.h"
#include "Account.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    //Створення об'єкту класу myAccount
    Account* myAccount = new Account("ztl@gmail.com","123qap&");

    //Створення двох покажчиків
    SmartPointer<Account> firstPointer(myAccount);
    SmartPointer<Account> secondPointer;
```

```
    if (!firstPointer.isNull()) {
        cout << " ---1. Друк інформації(перший покажчик)---\n";
        firstPointer->printInformation();
    }

    if (!secondPointer.isNull()) {
        cout << " ---1. Друк інформації(другий покажчик)---\n";
        secondPointer->printInformation();
    }

    //Передача управління об'єктом іншому покажчику
    secondPointer = firstPointer;

    if (!firstPointer.isNull()) {
        cout << " ---2. Друк інформації(перший покажчик)---\n";
        firstPointer->printInformation();
    }

    if (!secondPointer.isNull()) {
        cout << " ---2. Друк інформації(другий покажчик)---\n";
        secondPointer->printInformation();
    }
}
```

5. Створити клас *Human*. Поля класу:

– прізвище;

– вік.

В класі наявний віртуальний метод друку інформації.

Створити клас *Employee*, успадкований від *Human*. Полем класу є заробітна плата. Методи класу *Emploee*:

– сетер для поля класу;

– перевизначення віртуальної функції.

Створити клас *Teacher*, успадкований від *Employee*. Полями класу є:

– номер школи;

– предмет.

У класі наявне перевизначення віртуальної функції.

Створити шаблонний клас *Container*. Клас реалізовує зручний інтерфейс для роботи з колекцією динамічно створених екземплярів класу. Поля класу *Container*:

– масив покажчиків(тип шаблону);

– кількість елементів в масиві.

Методи класу:

– конструктор за замовчуванням;

– деструктор;

– перевантаження оператора(+=), для додавання нового об'єкту;

– перевантаження оператора квадратних дужок;

– функція вилучення покажчика з масиву;

– гетер для розміру масиву.

Реалізувати шаблонний клас у двох файлах:

– заголовний файл;

– файл реалізації.

При додаванні і вилученні елемента із масиву перестворювати його з новим розміром. Передбачити виняткові випадки. Явно оголосити три шаблони з типами даних:

– *human*;

– *employee*;

– *teacher*.

Розв'язання задачі 5:

Human.h

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

class Human
{
protected:

	//Прізвище
	string surname;

	//Вік
	int age;
```

```
public:

	//Конструктор з параметром
	Human(const string& surname,
		const int age);

	//Віртуальна функція для друку інформації
	virtual void printInfo() const;
};
```

Human.cpp

```
#include "Human.h"

//Конструктор з параметром
Human::Human(const string& surname, const int age)
{
	this->surname = surname;
	this->age = age;
}

//Віртуальна функція для друку інформації
void Human::printInfo() const
{
	cout << "Прізвище: " << surname
		<< "\nВік: " << age << endl;
}
```

Employee.h

```
#include "Human.h"

class Employee : public Human
{
protected:

	//Заробітна плата
	int salary;

public:

	//Конструктор з параметром
	Employee(const string& surname,
```

```
                const int age, const int salary);

        //Гетер для поля класу
        void setSalary(const int salary);

        //Перевизначення віртуальної функції
        virtual void printInfo() const override;
};
```

Employee.cpp

```
#include "Employee.h"

//Конструктор з параметром
Employee::Employee(const string& surname,
        const int age, const int salary) :
        Human(surname, age)
{
        this->salary = salary;
}

//Гетер для поля класу
void Employee::setSalary(const int salary)
{
        this->salary = salary;
}

//Перевизначення віртуальної функції
void Employee::printInfo() const
{
        Human::printInfo();
        cout << "Заробітня плата: " << salary << endl;
}
```

Teacher.h

```
#include "Employee.h"

class Teacher : public Employee
{
protected:

        //Номер школи
        int schoolNumber;
```

```
        //Предмет
        string subject;

public:

        //Конструктор з параметром
        Teacher(const string& surname,
                const int age, const int salary,
                const int schoolNumber,
                const string& subject);

        //Перевизначення віртуальної функції
        virtual void printInfo() const override;
};
```

Teacher.cpp

```
#include "Teacher.h"

//Конструктор з параметром
Teacher::Teacher(const string& surname, const int age,
                 const int salary, const int schoolNumber,
                 const string& subject):
                 Employee(surname, age, salary)
{
        this->schoolNumber = schoolNumber;
        this->subject = subject;
}

//Перевизначення віртуальної функції
void Teacher::printInfo() const
{
        Employee::printInfo();
        cout << "Номер школи: " << schoolNumber
               << "\nПредмет: " << subject << endl;
}
```

Container.h

```
#include "Teacher.h"

template<typename Type>
class Container
```

```
{
private:

	//Масив покажчиків
	Type** array;

	//Розмір масиву
	int arraySize;

public:

	//Конструктор за замовчуванням
	Container();

	//Деструктор
	~Container();

	//Перевантаження оператора(+=)
	Container<Type>& operator+=(Type* newItem);

	//Перевантаження оператора квадратних дужок
	Type& operator[](const int index);

	//Функція вилучення елемента по індексу
	Type* getElement(const int index);

	//Гетер для довжини контейнера
	int getSize() const;
};
```

Container.cpp

```
#include "Container.h"

//Явне оголошення шаблонів
template Container <Human>;
template Container <Employee>;
template Container <Teacher>;

//Конструктор за замовчуванням
template<typename Type>
inline Container<Type>::Container()
{
	this->array = nullptr;
	this->arraySize = 0;
```

```
}

//Деструктор
template<typename Type>
Container<Type>::~Container()
{

	//Якщо контейнер не пустий, звільнюємо пам'ять
	if (arraySize != 0) {

		//Звільнення пам'яті об'єктів
		for (int i = 0; i < arraySize; i++)
		{
			delete *(array + i);
		}

		//Звільнення пам'яті динамічного масиву
		delete[] array;
	}
}

//Перевантаження оператора(+=)
template<typename Type>
Container<Type>& Container<Type>::operator+=(Type* newItem)
{

	//Тимчасовий масив
	Type** tempArray = new Type* [arraySize + 1];

	//Перезапис масиву з додаванням нового елемента
	for (int i = 0; i < arraySize; i++)
	{
		*(tempArray + i) = *(array + i);
	}
	*(tempArray + arraySize) = newItem;

	//Змінюємо розмір масиву
	arraySize++;

	if (array != nullptr)
		delete[] array;

	//Переставляємо покажчик на масив
	array = tempArray;
```

```
	//Якщо ланцюг додавань
	return *this;
}

//Перевантаження оператора квадратних дужок
template<typename Type>
Type& Container<Type>::operator[](const int index)
{
	if (index >= arraySize && index < 0)
		throw exception("Недоступний індекс!");

	return **(array + index);
}

//Функція вилучення елемента по індексу
template<typename Type>
Type* Container<Type>::getElement(const int index)
{
	if (arraySize == 0)
		throw exception("Контейнер пустий!");

	if (index >= arraySize && index < 0)
		throw exception("Недоступний індекс!");

	//Запам'ятовуємо адрес елемента
	Type* returnedElement = *(array + index);

	//Якщо розмір масиву не дорівнює 1, є можливість
	//створити масив на один елемент менший
	if (arraySize != 1) {
		Type** tempArray = new Type* [arraySize - 1];

		//Перезаписуємо всі елементи масиву
		//крім видаляємого
		for (int i = 0, j= 0; i<arraySize - 1; i++, j++)
		{
			if (i == index) {
				j++;
			}
			*(tempArray + i) = *(array + j);
		}

		//Звільнення пам'яті
		delete[] array;
```

```
            //Переставляємо покажчик
            array = tempArray;
        }
        else {

            //Якщо розмір масиву дорівнює 1
            //видаляємо його та занулюємо
            delete[] array;
            array = nullptr;
        }

        //Зменшення розміру масиву
        arraySize--;

        //Повернення покажчика на об'єкт
        return returnedElement;
}

//Гетер для довжини контейнера
template<typename Type>
int Container<Type>::getSize() const
{
        return this->arraySize;
}
```

Practice_task_5.cpp

```
#include <Windows.h>
#include "Container.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Container<Human> container;

    container += new Human("Мельников", 23);
    container += new Employee("Паржницький", 19, 1000);
    container += new Teacher("Пишкін",30,12000,15,"Математика");

    try
    {
        cout << " --- Друк контейнера ---\n";
        for (int i = 0; i < container.getSize(); i++)
```

```
        {
            cout << " - Інформація про " << i + 1
                << " людину - " << endl;
            container[i].printInfo();
            cout << endl;
        }

        //Передача управління пам'яттю покажчику
        Human* pointer = container.getElement(1);

        cout << " --- Оновлений контейнер ---" << endl;
        for (int i = 0; i < container.getSize(); i++)
        {
            cout << " - Інформація про " << i + 1
                << " людину - " << endl;
            container[i].printInfo();
            cout << endl;
        }

        cout << "---Друк інформації за допомогою покажчика---n";
        pointer->printInfo();

        //Звільняємо пам'ять
        delete pointer;
    }
    catch (const exception& exception)
    {
        cout << exception.what() << endl;
    }
}
```

Контрольні запитання

1. Що таке шаблон? Для чого їх використовують?

2. За допомогою якого ключового слова оголошується шаблон функції чи класу?

3. Скільки узагальнених типів може бути у шаблоні?

4. Описати синтаксичну конструкцію створення екземпляра шаблонного класу.

5. Опишіть особливості створення шаблонних класів під час роздільної компіляції програми.

6. Чи можна успадковуватися від шаблонного класу?

9. БІБЛІОТЕКА СТАНДАРТНИХ МЕТОДІВ ТА КЛАСІВ STL

Мета: Оволодіння практичними навичками роботи з бібліотекою стандартних методів та класів STLв мові С++

***Теми для попереднього опрацьовування*:**

- Перевантаження операторів.
- Шаблонні функції.
- Шаблонні класи.
- Контейнерні класи.

Теоретичні відомості

STL (*standard template library*) – бібліотека стандартних методів і класів, що входить до складу будь-якої системи програмування, заснованою на мові С++. Ця бібліотека заснована на шаблонах.

До складу STL входять :

- потокові класи;
- класи для роботи з рядками(*string*);
- контейнерні класи;
- ітератори і алгоритми для роботи з контейнерними класами;
- математичні класи;
- діагностичні класи (в т.ч. клас *exception*).

Контейнерні класи реалізують найбільш поширені моделі обробки і зберігання даних. Вони призначені для зберігання однорідних даних.

Елементом контейнера можуть бути як стандартні типи, так і типи, визначувані користувачем. Якщо як елемент контейнера виступає визначений користувачем клас, цей клас повинен містити правильний конструктор без параметрів, конструктор копіювання і реалізацію операції привласнення.

Класифікація контейнерів

1. *Послідовні* контейнери (*vector*, *deque*, *list*) забезпечують зберігання однотипних величин у вигляді безперервної послідовності, тобто визначені поняття "початковий елемент", "кінцевий елемент", "попередній

елемент", "подальший елемент".

2. Використання *асоціативних* контейнерів (*map*, *multimap*, *set*, *multiset*, *bitset*) передбачає, що в даних, що зберігаються, виділяється ключ, який визначає конкретний елемент даних, що зберігаються, і неключова інформація. Асоціативні контейнери забезпечують швидкий доступ до даних по значеннях ключа.

3. *Адаптери* (*stack*, *queue*, *priority_queue*) реалізовані на основі базових послідовних контейнерів і призначені для виконання обмеженого набору операцій. Кожен адаптер будується з використанням механізму реалізації конкретного базового класу.

Створення контейнерів

Для створення контейнера необхідно як параметр шаблону вказати тип даних, що зберігаються в шаблоні :

```
vector <int> v;
```

Для створення адаптера, окрім типу даних, можна вказувати базовий контейнер, використовуваний для створення адаптера :

```
// базовий контейнер – vector <int>
stack <int, vector <int> > s2;
```

Стек працює за принципом «останній прийшов – перший пішов» (LIFO від англійського Last In – First Out).

Ітератор використовується для доступу до елементів контейнера. Ітератор є аналогом покажчика на елемент, хоча фізична природа ітератора може і не передбачати зберігання адрес. Все, що потрібний від ітератора – уміти посилатися на елемент контейнера і забезпечити перехід до іншого елемента (найчастіше до подальшого або до попереднього).

Ітератори діляться на *прямі* і *реверсивні*. Прямі ітератори використовуються при переходах від першого до останнього елемента контейнера, реверсивні – в зворотному порядку..

Ітератор описаний у кожному контейнері як допоміжний клас, тому для визначення ітератора необхідно вказувати контейнер, для якого ітератор буде використовуватися:

```
vector <int> :: iterator i1;
```

Можна зробити і так:

```
typedef vector <int> :: iterator it_vint;
```

```
it_vint i1, i2, i3;
```

Основні операції над ітераторами

Нехай *i* – ітератор, *n* – ціле число, *c* - контейнер.

Всі ітератори:

**i* – отримання доступу до елемента контейнера, з яким пов'язаний ітератор;

i++ – після цієї операції ітератор стає пов'язаним з наступним елементом контейнера;

i+*n* – переміщення ітератора на n елементів (при цьому можливий вихід за межі контейнера!). Для ітераторів, пов'язаних з контейнером *list*, ця операція недопустима;

*i*1 = *i*2 – привласнення;

*i*1 == *i*2, *i*1 != *i*2 – порівняння (інших операцій порівняння немає!);

i = *c.begin*() – ітератор зв'язується з першим елементом контейнера *c*;

i = *c.end*() – ітератор зв'язується з останнім елементом контейнера *c*.

Прямі ітератори:

i = *c.begin*() – ітератор зв'язується з першим елементом контейнера *c*;

i = *c.end*() – ітератор зв'язується з фіктивним елементом контейнера *c*, який «стоїть» за останнім елементом. Ітератор *i* стає недійсним!

Реверсивні ітератори:

i = *c.rbegin*() – ітератор зв'язується з останнім елементом контейнера *c*;

i = *c.rend*() – ітератор зв'язується з фіктивним елементом контейнера *c*, який «стоїть» перед першим елементом. Ітератор і стає недійсним!

***Приклад* 1**. Приклади роботи з ітераторами.

```
// перегляд всіх елементів списку
typedef list <int> :: iterator it_lint;
list <int> L;
for (int i = 1; i <= 10; i++)         // заповнення списку
        L.push_back(i);
                                      // друк списку
for (it_lint i = L.begin(); i!=L.end(); i++)
  cout  << *i << " ";

for (auto p = L.begin(); p != L.end(); p++)
        cout << setw(4) << *p;
```

```
for(const auto &x :L) cout << x << " ";

    // перегляд всіх елементів списку в зворотному порядку
typedef list <int> :: reverse_iterator rit_lint;
for (rit_lint i = L.rbegin(); i!=L.rend(); i++)
  cout  << *i << endl;

for (auto p = L.rbegin(); p != L.rend(); p++)
          cout << setw(4) << *p;
```

Послідовні контейнери. Ініціалізація

Спосіб ініціалізації забезпечується *конструкторами* контейнерних класів.

1. Конструктор порожнього вектору (К за умовчанням)

```
vector <int> v1;
```

2. Конструктор з параметрами

```
vector <int> v2(5); // вектор v2 міститиме 5 елементів,
                    // привласнюватися значення будуть пізніші
vector <int> v3(5,8); // п'ять елементів, все = рівні 8
```

3. Конструктор копіювання

```
vector <int> v4(v3);
vector <int> v5(v4.begin()+1,v4.end()-1);
```

4. Ініціалізація елементами масиву

```
int a[3] = {10, 5, 7};
vector<int> v6(a, a+3);
```

Наступні *операції* допустимі над всіма *послідовними* контейнерами:

– вставка в початок контейнера (перед першим елементом) – *push_front*;

– вставка в кінець контейнера (після останнього елемента)– *push_back*;

– видалення початкового елемента – *pop_front*;

– видалення останнього елемента – *pop_back*;

– вставка в довільне місце – *insert*;

– видалення з довільного місця – *erase*;

– доступ до елемента по його індексу – *at* або [ ].

***Приклад* 2.** Приклади роботи з контейнером *vector*.

```
// описуємо два вектори
   vector <int> v1, v2;
// заповнення v1: 1 2 3 4 5 6 7 8 9 10
  for (int i=1; i<=10; i++) v1.push_back(i);
// друк v1
   for (const auto& x : v1)
      cout << x << " ";
  copy(v1.begin(), v1.end(),ostream_iterator<int>(cout," "));

// вставка в v2
  v2.insert(v2.begin(), 15);          // v2: 15
  v2.insert(v2.end(), 5, 2);          // v2: 15 2 2 2 2 2
  v2.insert(v2.begin(), v1.begin()+5, v1.begin()+7);
                                   // v2: 6 7 15 2 2 2 2 2

//видалення з v1
  v1.erase(v1.begin());          // v1: 2 3 4 5 6 7 8 9 10
  v1.erase(v1.begin()+4, v1.end()-1);    // v1: 2 3 4 5 10
```

***Приклад* 3.** Приклади роботи з контейнером *vector*. Видалити з контейнера всі елементи, менше 5.

```
vector <int> v1;
for (int i = 1; i <= 10; i++)
   v1.push_back(i);
vector <int> ::iterator it;   // auto it = v1.begin();
for (int i = 0; i < v1.size();  )
{
  if (v1[i] < 5)
  {
    it = (v1.begin() + i);
    v1.erase(it);
  }
  else
    i++;
}
```

```
// Інший варіант
vector <int>::iterator i;
for (i = v1.begin(); i != v1.end();   )
{
  if (*i < 5)
    i = v1.erase(i); // erase повертає правильний
                     // ітератор на елемент,
                     // наступний за видаленим
  else i++;
}
```

***Приклад* 4.** Приклади роботи з контейнером *list*. Зробити функцію друку списку.

```
typedef list <int> :: iterator it_lint;
list <int> L(10), L1;

// функція друку
void WriteList(list <int> L)
{
  cout << L.size() << ": ";
  for (it_lint i = L.begin(); i != L.end(); i++)
    cout  << *i << " ";
 // або for (const auto& x : L) cout << x << " ";
  cout << endl;
}

int c = 0;
for (auto i = L.begin(); i != L.end(); i++)
  (*i) = (c += 4);
WriteList(L);           // 10: 4 8 12 16 20 24 28 32 36 40

// Вставляє перед вказаною позицією всі
// елементи із списку-джерела; джерело очищається
L1.splice(L1.begin(), L);
WriteList(L);           // 0:
WriteList(L1);          // 10: 4 8 12 16 20 24 28 32 36 40
```

```
// те ж, але із списку-джерела переноситься
// лише один елемент
L1.splice(L1.begin(), L, L.begin());
WriteList(L);           // 9: 8 12 16 20 24 28 32 36 40
WriteList(L1);          // 1: 4
```

Асоціативні контейнери

Використання асоціативних контейнерів передбачає, що в даних, що зберігаються, виділяється ключ, який визначає конкретний елемент даних, що зберігаються, і значення по ключу. Асоціативні контейнери забезпечують швидкий доступ до даних по значеннях ключа.

Основні *операції* для асоціативних контейнерів:

- вставка елемента;
- видалення елемента;
- пошук елемента по ключу;
- послідовний перегляд (в порядку зростання ключів, незалежно від порядку вставок).

Допоміжний клас pair

Це просто шаблонна структура, яка містить два поля, можливо, різних типів. Поля мають назви *first* і *second*.

Прототип пари може виглядати таким чином:

```
template <typename T1, typename T2>
   class pair {
     T1 first;
     T2 second;
  };

pair <string, pair <int, int> > P; // рядок і два
                                   // цілих числа
    string s = P.first;      // Рядок
    int x = P.second.first;  // Перше ціле
    int y = P.second.second; // Друге ціле
```

Контейнери set та multiset (множини)

Множина – ***set***. Кожен елемент множини є власним ключем, і ці ключі унікальні. Тому два різні елементи множини не можуть збігатися. Наприклад, множина може складатися з наступних елементів:

```
123    124    800    950
```

Множина з дублікатами – ***multiset***. Множина з дублікатами відрізняється від просто множини лише тим, що здатна містити декілька співпадаючих елементів:

```
123    123    800    950
```

На відміну від послідовних контейнерів асоціативні контейнери зберігають свої елементи *відсортованими*, незалежно від того, яким чином вони були додані.

***Приклад* 5.** Приклади роботи з асоціативними контейнерами.

```
set<int> S;
S.insert(20);
S.insert(10);
S.insert(30);
S.insert(10);
for (const auto& x : S)
  cout << x << " ";           // 10 20 30

multiset<int> T;
T.insert(20);
T.insert(30);
T.insert(10);
T.insert(10);
for (const auto& x : T)
  cout << x << " ";           // 10 10 20 30
```

***Приклад* 6.** Приклади роботи з асоціативними контейнерами.

```
// Предикат  less<int>
// для впорядкування за збільшенням (за умовчанням)
set <int,less<int> > S;
S.insert(10);
S.insert(20);
S.insert(30);
```

```
S.insert(10);
for (const auto& x : S)
  cout << x << " ";            // 10 20 30


// Предикат greater<int>
// для впорядкування по убуванню
multiset <int, greater<int>> T;
T.insert(20);
T.insert(30);
T.insert(10);
T.insert(10);
for (const auto& x : T)
  cout << x << " ";            // 30 20 10 10
```

***Приклад* 7.** Приклади роботи з асоціативними контейнерами. Реалізація телефонної книги контейнером *map*.

Ключ: інформація про власника контакту (ПІП)

Значення по ключу : інформація про контакт (номер телефону)

```
map <string, string> Phb;
map <string, string> ::iterator it;
```

В класі *map* перевизначена операція взяття індексу:

```
T& operator[](const Key&)
```

так, що за допомогою цієї операції можна здійснювати пошук даних по ключу або змінювати вміст неключових даних. Ця ж операція дає можливість вставляти новий елемент контейнера.

```
// Вставити новий елемент з ключем "kash"
// і номером телефону "123456789"
Phb["kash"] = "123456789";

cout << Phb["kash"] << endl;          // 123456789
cout << Phb.size() << endl;           // 1
for (const auto& x : Phb)
  cout << x.first << " " << x.second << endl;
                                      // kash 123456789
```

```
cout << Phb["pupkin"] << endl;
// Результат – порожній рядок, оскільки ключ не знайдений.
// Проте в словник буде вставлений новий елемент!
cout << Phb.size() << endl;           //2
 for (const auto& x : Phb)
        cout << x.first << " " << x.second << endl;
                                      // kash 123456789
                                      // pupkin

// Знайти телефон по прізвищу або видати повідомлення
// про відсутність даних.
    string N1;
    cout << "Enter name: ";           // Enter name: kash
    cin >> N1;
   it = Phb.find(N1);
    cout << (it == Phb.end() ?
         "Record not found" : (*it).second) << endl;
                                      // 123456789


// Вставити новий елемент по парі «ключ – неключові дані».
    Phb.insert({ "kirill", "2353555" });
  for (const auto& x : Phb)
        cout << x.first << " " << x.second << endl;
                        // kash 123456789
                        // kirill 2353555
                        // pupkin

// Видати вміст телефонної книги з іменами,
// що починаються на "k". lower_bound() повертає ітератор,
// вказуючий на першу позицію у відсортованій
// послідовності, обмеженій діапазоном [first,last],
 for(it = Phb.lower_bound("k");
          it != Phb.lower_bound("l"); it++)
{
    cout << it -> first << ":" << (*it).second << endl;
}                       // kash:123456789
                        // kirill:2353555
```

```
// Видалити всі записи з порожніми номерами телефон.
   map <string, string> ::iterator jt;
for (it = Phb.begin(); it != Phb.end();  )
{
    jt= it;
    it++;
    if ((*jt).second.length() == 0)
    {
       cout << "Deleting: " << (*jt).first << endl;
       Phb.erase(jt);
    }
}
for (const auto& x : Phb)
        cout << x.first << " " << x.second << endl;
                                // Deleting: pupkin
```

***Приклад* 8.** Приклади роботи з асоціативними контейнерами. Реалізація телефонної книги контейнером *multimap*.

Ключ: інформація про власника контакту (ПІП)

Значення по ключу : інформація про контакт (номер телефону)

```
multimap <string, string> Phb;
multimap <string, string> ::iterator it;
```

Клас *miltimap* допускає зберігання декількох елементів з однаковими ключами (допускається також повне дублювання елементів, що зберігаються). Тому в цьому класі операція доступу по індексу заборонена, і замість неї треба користуватися ітераторами.

```
// Phb["kash"] = "123456789";  //не працює

Phb.insert(pair<string, string>("kash", "111111111"));
Phb.insert(pair<string, string>("kash", "222222222"));
Phb.insert({"kash", "333333333"});
Phb.insert({ "kirill", "444444444" });

for (it = Phb.begin(); it != Phb.end(); it++)
```

```
      cout << it->first << " " << it->second << endl;
// Вивести всю інформацію, що зберігається, для ключа
//"kash". Звернення до equal_range() із значенням kash
// повертає пару ітераторів (у діапазоні - початковий і
// кінцевий), у якій обидва вказують на значення kash

    pair < multimap <string, string> ::iterator,
           multimap <string, string> ::iterator>  p_it;

    p_it = Phb.equal_range("kash");

 for (it = p_it.first; it != p_it.second; it++)
       cout << (*it).first << ":" << (*it).second <<endl;
                                // kash:111111111
                                // kash:222222222
                                // kash:333333333

//Виправити номер для одного елемента ключа "kash“.
for (it = p_it.first; it != p_it.second; it++)
{
        if ((*it).second == "111111111")
       {
            (*it).second = "999999999";
            break;
        }
}
    for (it = p_it.first; it != p_it.second; it++)
      cout << it -> first << ":" << it -> second <<endl
                                // kash:999999999
                                // kash:222222222
                                // kash:333333333
```

Задачі

Загальні умови

Кожне завдання містить приклад використання бібліотеки стандартних методів та класів STL у мові C++, і може містити умови вирішення. Після формулювання завдання надано рішення у вигляді програмного коду.

1. Створити клас *Date* з полями:

– день;

– місяць;

– рік.

Методами класу *Date* є:

– конструктор з параметрами за замовчуванням;

– конструктор копіювання;

– гетер для поля року;

– перевантаження оператора(<<);

– перевантаження оператора(=);

– перевантаження оператора(==).

Створити клас *Person*, полями класу є:

– ім'я;

– дата народження(*Date*);

Методи класу *Person*:

– конструктор з параметрами за замовчуванням;

– конструктор копіювання;

– група "гетерів" класу;

– перевантаження оператора(<<);

– перевантаження оператора(<), який порівнює рік народження;

– перевантаження оператора(=).

Створити клас *BirthdayList*, у якому зберігаються потрібні користувачу дати днів народження. Полем класу є контейнер *multiset* з типом *Person*. Методи класу:

– функція додавання людини;

– функція видалення людини за датою народження;

– функція друку списку;

– функція пошуку людини у списку(по імені), повертає ітератор.

Розв'язання задачі 1:

Date.h

```
#include <iostream>
```

```
using namespace std;

class Date
{
private:

	//День
	int day;

	//Місяць
	int month;

	//Рік
	int year;

public:

	//Конструктор з параметрами за замовчуванням
	Date(const int day = 1, const int month = 1,
	     const int year = 1);

	//Конструктор копіювання
	Date(const Date& date);

	//Гетер для року
	int getYear() const;

	//Перевантаження оператора(<<)
	friend ostream& operator<<(ostream& out,
	                           const Date& date);

	//Перевантаження оператора(=)
	Date& operator=(const Date& date);

	//Перевантаження оператора(==)
	friend bool operator==(const Date& firstDate,
	                       const Date& secondDate);
};
```

Date.cpp

```
#include "Date.h"

//Конструктор з параметрами за замовчуванням
Date::Date(const int day, const int month, const int year)
```

```
{
	this->day = day;
	this->month = month;
	this->year = year;
}

//Конструктор копіювання
Date::Date(const Date& date)
{
	this->day = date.day;
	this->month = date.month;
	this->year = date.year;
}

//Перевантаження оператора(=)
Date& Date::operator=(const Date& date)
{
	this->day = date.day;
	this->month = date.month;
	this->year = date.year;

	return *this;
}

//Гетер для року
int Date::getYear() const
{
	return this->year;
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const Date& date)
{
	return out << date.day << "." << date.month
		<< "." << date.year;
}

//Перевантаження оператора(==)
bool operator==(const Date& firstDate, const Date& secondDate)
{
	return (firstDate.day == secondDate.day)
		 && (firstDate.month == secondDate.month)
		 && (firstDate.year == secondDate.year);
}
```

Person.h

```
#include "Date.h"

class Person
{
private:

	//Прізвище
	string secondName;

	//День народження
	Date birthday;

public:

	//Конструктор з параметрами за замовчуванням
	Person(const string& secondName = "", const int day = 1,
		const int month = 1, const int year = 1);

	//Конструктор копіювання
	Person(const Person& person);

	//Група "гетерів" класу
	string getSecondName() const;
	Date getBirthday() const;

	//Перевантаження оператора(=)
	Person& operator=(const Person& person);

	//Перевантаження оператора(<)
	bool operator<(const Person& person) const;

	//Перевантаження оператора(<<)
	friend ostream& operator<<(ostream& out,
				const Person& person);
};
```

Person.cpp

```
#include "Person.h"

//Конструктор з параметрами за замовчуванням
Person::Person(const string& secondName, const int day,
```

```
                const int month, const int year)
{
	this->secondName = secondName;
	this->birthday = Date(day, month, year);
}

//Конструктор копіювання
Person::Person(const Person& person)
{
	this->secondName = person.secondName;
	this->birthday = person.birthday;
}

//Перевантаження оператора(=)
Person& Person::operator=(const Person& person)
{
	this->secondName = person.secondName;
	this->birthday = person.birthday;

	return *this;
}

//Перевантаження оператора(<)
bool Person::operator<(const Person& person) const
{
	return this->birthday.getYear()
			< person.birthday.getYear();
}

//Група "гетерів" класу
string Person::getSecondName() const
{
	return this->secondName;
}

Date Person::getBirthday() const
{
	return this->birthday;
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const Person& person)
{
	return out << person.secondName << "\t"
		<< person.birthday;
```

```
}
```

BirthdayList.h

```
#include "Person.h"
#include <set>

//Створення псевдоніма для ітератора
typedef multiset<Person>::iterator multiset_iterator;

class BirthdayList
{
private:

	multiset<Person> birthdayList;

public:

	//Функція додавання людини до списку
	void addPerson(const Person& newPerson);

	//Функція видалення людини зі списку
	bool deleteByBirthday(const Date& birthday);

	//Друк списку
	void printList() const;

	//Функція пошуку людини у списку
	multiset_iterator searchPerson(const string&
                                        secondName);
};
```

BirthdayList.cpp

```
#include "BirthdayList.h"

//Функція додавання людини до списку
void BirthdayList::addPerson(const Person& newPerson)
{
	birthdayList.insert(newPerson);
}

//Функція видалення людини зі списку
bool BirthdayList::deleteByBirthday(const Date& birthday)
```

```
{
	bool flag = 0;

	//Ставимо покажчик на початок multiset, ітеруємо
	//до кінця
	//Можна використовувати створений псевдонім або auto
	for (auto pMultiset = birthdayList.begin();
		  pMultiset != birthdayList.end(); pMultiset++)
	{

		//Якщо дати збіглися, видаляємо
		if (pMultiset->getBirthday() == birthday) {
			birthdayList.erase(pMultiset);
			flag = 1;
			break;
		}
	}

	//прапорець видалення елемента
	return flag;
}

//Друк списку
void BirthdayList::printList() const
{

	if (birthdayList.empty())
		throw exception("Список порожній!");

	//Ставимо покажчик на початок multiset, ітеруємо
	//до кінця
	for (auto pMultiset = birthdayList.begin();
		  pMultiset != birthdayList.end(); pMultiset++)
	{

		//Розіменовуємо покажчик, друкуємо інформацію
		cout << *pMultiset << endl;
	}
}

//Функція пошуку людини у списку
multiset_iterator BirthdayList::searchPerson
                                    (const string& secondName)
{
```

```
        for (auto pMultiset = birthdayList.begin();;
                                                pMultiset++)
        {

                //Вихід, якщо кінець
                if (pMultiset == birthdayList.end()) {
                        throw exception("Елемент не знайдено!");
                }

                //Якщо знайшли, повертаємо покажчик на елемент
                if(pMultiset->getSecondName() == secondName){
                        return pMultiset;
                }
        }
}
```

Practice_task_1.cpp

```
#include <Windows.h>
#include "BirthdayList.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    BirthdayList birthdayList;
    multiset_iterator msIterator;
    bool deleteFlag = 0;

    try
    {

        cout <<"--Створення та друк списку днів народження--\n";
        birthdayList.addPerson(Person("Паржницький",
                                      6, 11, 2003));
        birthdayList.addPerson(Person("Агоян",
                                      5, 9, 1999));
        birthdayList.addPerson(Person("Михайленко ",
                                      29, 1, 2004));
        birthdayList.addPerson(Person("Пишкін",
                                      22, 5, 1989));

        birthdayList.printList();
```

```
        //Пошук елемента
        msIterator = birthdayList.searchPerson("Агоян");

        cout << "\n-- Друк інформації через покажчик --\n";
        cout << *msIterator << endl;

        //Видалення елемента
        deleteFlag = birthdayList.
                     deleteByBirthday(Date(6, 11, 2003));

        if (deleteFlag) {
            cout << "\nЕлемент видалено!" << endl;
        }
        else {
            cout << "\nЕлемент не знайдено!" << endl;
        }

        cout << "\n-- Друк оновленого списку --\n";
        birthdayList.printList();

    }
    catch (const exception& exception)
    {
        cout << exception.what() << endl;
    }
}
```

2. Створити клас *GemstoneInfo*. Полями класу є:

– назва дорогоцінного каменю;

– місце знаходження.

Методи класу:

– конструктор з параметрами за замовчуванням;

– конструктор копіювання;

– перевантаження оператора(=);

– перевантаження оператора(<<).

Створити шаблонний клас *Collection*. Полем класу є контейнер *multimap*, де ключ - ціна(*int*), значення - інформація(довільний тип). Методи класу *Collection*:

– порожній конструктор за замовчуванням;

– конструктор копіювання;

– функція додавання елемента;

– функція видалення елемента за ключом;

– функція очищення контейнера;

– функція друку інформації;

– перевантаження оператора(+) для об'єднання колекцій;

– перевантаження оператора(=).

Явно оголосити шаблон класу з типом даних *GemstoneInfo*.

Розв'язання задачі 2:

GemstoneInfo.h

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

class GemstoneInfo
{
private:

	//Назва дорогоцінного каменю
	string nameOfGem;

	//Місце знаходження
	string location;

public:

	//Конструктор з параметрами за замовчуванням
	GemstoneInfo(const string& nameOfGem = "",
		     const string& location = "");

	//Конструктор копіювання
	GemstoneInfo(const GemstoneInfo& gemstone);

	//Перевантаження оператора(=)
	GemstoneInfo& operator=(const GemstoneInfo& gemstone);

	//Перевантаження оператора(<<)
	friend ostream& operator<<(ostream& out,
				   const GemstoneInfo& gemstone);
};
```

GemstoneInfo.cpp

```
#include "GemstoneInfo.h"

//Конструктор з параметрами за замовчуванням
GemstoneInfo::GemstoneInfo(const string& nameOfGem,
                           const string& location)
{
	this->nameOfGem = nameOfGem;
	this->location = location;
}

//Конструктор копіювання
GemstoneInfo::GemstoneInfo(const GemstoneInfo& gemstone)
{
	this->nameOfGem = gemstone.nameOfGem;
	this->location = gemstone.location;
}

//Перевантаження оператора(=)
GemstoneInfo& GemstoneInfo::operator=
                              (const GemstoneInfo& gemstone)
{
	this->nameOfGem = gemstone.nameOfGem;
	this->location = gemstone.location;

	return *this;
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const GemstoneInfo& gemstone)
{

	out << "Назва: " << setw(10) << gemstone.nameOfGem
	    << " | Знаходження: " << setw(14)
	    << gemstone.location;

	return out;
}
```

Collection.h

```
#include <map>
#include "GemstoneInfo.h"
```

```
template<typename Type>
class Collection
{
private:

	//Контейнер multimap, ключ - ціна,
	//значення - об'єкт з довільним
	//типом даних
	multimap<int, Type> container;

public:

	//Конструктор за замовчуванням
	Collection();

	//Конструктор копіювання
	Collection(const Collection<Type>& object);

	//Функція додавання елемента
	void add(const int price, const Type& newElement);

	//Функція видалення елемента
	bool remove(const int price);

	//Функція очищення контейнера
	void clear();

	//Функція друку інформації
	void printCollection() const;

	//Перевантаження оператора(=)
	Collection<Type>& operator=
				(const Collection<Type>& object);

	//Перевантаження оператора(+)
	Collection<Type> operator+
				(const Collection<Type>& object);
};
```

Collection.cpp

```
#include "Collection.h"

//Явне оголошення шаблону
template Collection<GemstoneInfo>;
```

```
//Конструктор за замовчуванням
//Потрібен для створення об'єкта класу
template<typename Type>
Collection<Type>::Collection()
{
}

//Конструктор копіювання
template<typename Type>
Collection<Type>::Collection(const Collection<Type>& object)
{
	this->container = object.container;
}

//Функція додавання елемента
//Ключем є ціна, значенням
//інформація про камінь
template<typename Type>
inline void Collection<Type>::add(const int price,
                                  const Type& newElement)
{
	this->container.insert({ price, newElement });
}

//Функція видалення елемента
template<typename Type>
bool Collection<Type>::remove(const int price)
{
	bool removeFlag = 0;

	//Якщо елемент наявний у колекції
	//видаляємо його, і змінюємо прапорець
	if (container.find(price) != container.end()) {

		container.erase(price);
		removeFlag = 1;
	}

	return removeFlag;
}

//Функція очищення контейнера
template<typename Type>
void Collection<Type>::clear()
```

```
{
	container.clear();
}

//Функція друку інформації
template<typename Type>
void Collection<Type>::printCollection() const
{
	if (container.empty())
		throw exception("Колекція порожня!");

	for (auto pMultimap = container.begin();
		pMultimap != container.end(); pMultimap++)
	{
		cout << "Ціна: " <<  left <<setw(8)
		     << pMultimap->first << " | "
		     << pMultimap->second << endl;
	}
}

//Перевантаження оператора(=)
template<typename Type>
Collection<Type>& Collection<Type>::operator=(const
                                    Collection<Type>& object)
{
	this->container = object.container;

	//Повертаємо поточний об'єкт, щоб мати можливість
	//зв'язати в ланцюжок виконання кількох операцій
	//присвоєння
	return *this;
}

//Перевантаження оператора(+)
template<typename Type>
Collection<Type> Collection<Type>::operator+(const
                                    Collection<Type>& object)
{

	for (auto pMultimap = object.container.begin();
		pMultimap != object.container.end(); pMultimap++)
	{
		this->add(pMultimap->first, pMultimap->second);
	}
```

```
        return *this;
}
```

Practice_task_2.cpp

```
#include <Windows.h>
#include "Collection.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    //Створення двох колекцій
    Collection<GemstoneInfo> firstCollection;
    Collection<GemstoneInfo> secondCollection;

    try
    {

       //Створення першої колекції
       firstCollection.add(10000,
                        GemstoneInfo("Ізумруд", "США"));
       firstCollection.add(10000,
                        GemstoneInfo("Бріліант", "Німеччина"));
       firstCollection.add(20000,
                        GemstoneInfo("Сапфір", "Україна"));

       //Створення другої колекції
       secondCollection.add(15000,
                           GemstoneInfo("Сапфір", "Швеція"));
       secondCollection.add(33000,
                           GemstoneInfo("Рубін","Данія"));

       cout << " --- Перша колекція ---\n";
       firstCollection.printCollection();

       cout << "\n--- Друга колекція ---\n";
       secondCollection.printCollection();

       //Об'єднання колекцій
       secondCollection = secondCollection + firstCollection;

       cout << "\n--- Оновлена друга колекція ---\n";
       secondCollection.printCollection();
```

```
        //Видалення всіх елементів з ключем 10000
        if (firstCollection.remove(10000)) {
            cout << "\nЕлемент видалено!" << endl;
        }
        else {
            cout << "\nЕлемент не знайдено!" << endl;
        }

        cout << "\n--- Оновлена перша колекція ---\n";
        firstCollection.printCollection();

        //Очищення колекцій
        firstCollection.clear();
        secondCollection.clear();

        cout << "\n--- Друк порожньої колекції ---\n";
        firstCollection.printCollection();
    }
    catch (const exception& exception)
    {
        cout << exception.what() << endl;
    }
}
```

3. Створити клас *FileInfo*. Полями класу є:

– шлях до файлу;

– рік створення файлу;

– кількість звернень до файлу.

Методи класу:

– конструктор з параметрами за замовчуванням;

– конструктор копіювання;

– група "гетерів";

– перевантаження оператора(<<);

– перевантаження оператора(>>);

– перевантаження оператора(=).

Створити клас *FileManager*, який реалізовує зручний інтерфейс для роботи з файлами. Полем класу є контейнер *list* з типом *FileInfo*. Методи класу:

– функція створення каталогу з файлу;

– додавання нового файлу до каталогу;

– запис каталогу до файлу;

– функція друку каталогу;

– функція видалення файлів, дата створення яких менша за задану;

– статична функція-предикат, яка порівнює рік створення файлів;

– функція сортування каталогу за роком створення файлів;

– функція пошуку файлу з найбільшою кількістю звернень, повертає ітератор на елемент.

Для демонстрації програми передбачити зручний інтерфейс.

Розв'язання задачі 3:

FileInfo.h

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

class FileInfo
{
private:

	//Шлях до файлу
	string path;

	//Рік створення файлу
	int creationYear;

	//Кількість звернень до файлу
	int requestsNumber;

public:

	//Конструктор з параметрами за замовчуванням
	FileInfo(const string& path = "",
		 const int creationYear = 0,
		  const int requestsNumber = 0);

	//Конструктор копіювання
	FileInfo(const FileInfo& file);

	//Група "гетерів" класу
```

```
        string getPath() const;
        int getCreationYear() const;
        int getRequestsNumber() const;

        //Перевантаження оператора(<<)
        friend ostream& operator<<(ostream& out,
                                   const FileInfo& file);

        //Перевантаження оператора(>>)
        friend istream& operator>>(istream& in, FileInfo& file);

        //Перевантаження оператора(=)
        FileInfo& operator=(const FileInfo& file);
};
```

FileInfo.cpp

```
#include "FileInfo.h"

//Конструктор з параметрами за замовчуванням
FileInfo::FileInfo(const string& path, const int creationYear,
        const int requestsNumber)
{
        this->path = path;
        this->creationYear = creationYear;
        this->requestsNumber = requestsNumber;
}

//Конструктор копіювання
FileInfo::FileInfo(const FileInfo& file)
{
        this->path = file.path;
        this->creationYear = file.creationYear;
        this->requestsNumber = file.requestsNumber;
}

//Група "гетерів" класу
string FileInfo::getPath() const
{
        return this->path;
}

int FileInfo::getCreationYear() const
{
        return this->creationYear;
```

```
}

int FileInfo::getRequestsNumber() const
{
	return this->requestsNumber;
}

//Перевантаження оператора(=)
FileInfo& FileInfo::operator=(const FileInfo& file)
{
	this->path = file.path;
	this->creationYear = file.creationYear;
	this->requestsNumber = file.requestsNumber;

	return *this;
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const FileInfo& file)
{
	out << file.path << " "
	    << file.creationYear << " "
	    << file.requestsNumber;

	return out;
}

//Перевантаження оператора(>>)
istream& operator>>(istream& in, FileInfo& file)
{
	return in >> file.path
	          >> file.creationYear
	          >> file.requestsNumber;
}
```

FileManager.h

```
#include <list>
#include <fstream>
#include "FileInfo.h"

//Псевдонім для ітератора
typedef list<FileInfo>::iterator catalog_it;

class FileManager
```

```
{
private:

	//Контейнер list для зберігання
	//інформації про файли
	list<FileInfo> fileCatalog;

public:

	//Функція створення каталогу з файлу
	void creationFromFile(const string& path);

	//Додавання нового файлу до каталогу
	void addFile(const FileInfo& newFile);

	//Запис каталогу до файлу
	void catalogToFile(const string& path);

	//Функція друку каталогу
	void printCatalog() const;

	//Функція видалення файлів, дата створення
	//яких менша за задану
	void removeElements(const int year);

	//Сортування каталогу
	void sortCatalog();

	//Статична функція-предикат
	//Потрібна для сортування каталогу
	static bool predicate(const FileInfo& firstFile,
	                      const FileInfo& secondFile);

	//Функція пошуку файлу з найбільшою
	//кількістю звернень, повертає ітератор
	catalog_it getMaxRequestsNumbFile();
};
```

FileManager.cpp

```
#include "FileManager.h"

//Функція створення каталогу з файлу
void FileManager::creationFromFile(const string& path)
{
```

```
	ifstream fin(path);

	if (!fin.is_open())
		throw exception("Помилка відкриття файлу!");

	if (!fileCatalog.empty())
		fileCatalog.clear();

	//Тимчасова змінна
	FileInfo tempFile;

	//Зчитування інформації з файлу
	while (!fin.eof())
	{
		fin >> tempFile;
		fileCatalog.push_back(tempFile);
	}

	fin.close();
}

//Додавання нового файлу до каталогу
void FileManager::addFile(const FileInfo& newFile)
{
	fileCatalog.push_back(newFile);
}

//Запис каталогу до файлу
void FileManager::catalogToFile(const string& path)
{
	ofstream fout(path);

	if (!fout.is_open())
		throw exception("Помилка відкриття файлу!");

	if (fileCatalog.empty())
		throw exception("Каталог порожній!");

	//Розмір каталогу
	int size = fileCatalog.size();

	//Лічильник
	int counter = 0;

	//Записуємо інформацію у файл
```

```
	for (auto pList = fileCatalog.begin();
		pList != fileCatalog.end(); pList++, counter++)
	{
		fout << *pList;

		//Якщо останній елемент каталогу
		//не робимо отступ у файлі
		if (counter != size - 1)
			fout << endl;
	}

	fout.close();
}


//Функція друку каталогу
void FileManager::printCatalog() const
{
	if (fileCatalog.empty())
		throw exception("Каталог порожній!");

	for (auto pList = fileCatalog.begin();
		   pList != fileCatalog.end(); pList++)
	{
		cout << "Шлях: " << pList->getPath()
		<< " | Рік створення: "
		<< pList->getCreationYear()
		<< " | Кількість звернень: "
		<< pList->getRequestsNumber()
		<< endl;
	}
}

//Функція видалення файлів, дата створення
//яких менша за задану
void FileManager::removeElements(const int year)
{
	if (fileCatalog.empty())
		throw exception("Каталог порожній!");


	for (auto pList = fileCatalog.begin();
		   pList != fileCatalog.end();)
	{
```

```
                //Якщо рік збігся видаляємо елемент і
                //оновлюємо ітератор
                if(pList->getCreationYear() < year){

                        pList = fileCatalog.erase(pList);
                }
                else {
                        pList++;
                }
        }
}

//Статична функція-предикат
//Потрібна для сортування каталогу
bool FileManager::predicate(const FileInfo& firstFile,
                            const FileInfo& secondFile)
{
        return firstFile.getCreationYear()
                        > secondFile.getCreationYear();
}

//Сортування каталогу
void FileManager::sortCatalog()
{
        if (fileCatalog.empty())
                throw exception("Каталог порожній!");

        fileCatalog.sort(FileManager::predicate);
}

//Функція пошуку файлу з найбільшою
//кількістю звернень, повертає ітератор
catalog_it FileManager::getMaxRequestsNumbFile()
{
        if (fileCatalog.empty())
                throw exception("Каталог порожній!");

        //Покажчик на елемент, що повертається
        catalog_it returnedFile = fileCatalog.begin();

        for (auto pList = fileCatalog.begin();
                  pList != fileCatalog.end(); pList++)
        {

                //Якщо умова повертає true переставляємо покажчик
```

```
            if (pList->getRequestsNumber()
                > returnedFile->getRequestsNumber()) {
                   returnedFile = pList;
            }
        }

        return returnedFile;
}
```

Practice_task_3.cpp

```
#include <Windows.h>
#include "FileManager.h"

int main()
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);

        FileManager fileManager;
        catalog_it iterator;

        int auxiliaryFirstNumber = 0;
        int auxiliarySecondNumber = 0;
        string auxiliaryStr;

        while (true)
        {
                cout << "\n1. Заповнення з файлу" << endl;
                cout << "2. Додати файл" << endl;
                cout << "3. Записати каталог у файл" << endl;
                cout << "4. Надрукувати каталог" << endl;
                cout << "5. Видалити файли, дата створення"
                     << "яких менша за задану" << endl;
                cout << "6. Сортувати каталог за датою створення"
                     << endl;
                cout << "7. Надрукувати файл з найбільшою"
                     << "кількістю зверненнь" << endl;
                cout << " -> "; cin >> auxiliaryFirstNumber;

                try
                {
                        switch (auxiliaryFirstNumber)
                        {
                        case 1:
```

```
  cout << "Введіть шлях до файлу: ";
  cin >> auxiliaryStr;

  fileManager.creationFromFile
                      (auxiliaryStr);
       break;
case 2:
  cout << "Введіть(шлях, рік створення,"
       << "кількість звернень): ";
   cin >> auxiliaryStr
       >> auxiliaryFirstNumber
       >> auxiliarySecondNumber;

  fileManager.addFile(FileInfo(
  auxiliaryStr,
  auxiliaryFirstNumber,
  auxiliarySecondNumber));
  break;
case 3:
  cout << "Введіть шлях до файлу: ";
  cin >> auxiliaryStr;

  fileManager.catalogToFile(auxiliaryStr);
  break;
case 4:
  fileManager.printCatalog();
  break;
case 5:
  cout << "Введіть рік: ";
  cin >> auxiliaryFirstNumber;

  fileManager.removeElements
              (auxiliaryFirstNumber);
       break;
case 6:
  fileManager.sortCatalog();
  cout << "Каталог відсортовано" << endl;
  break;
case 7:
  iterator = fileManager.
             getMaxRequestsNumbFile();

  cout << *iterator << endl;
  break;
default:
```

```
                        exit(0);
                        break;
                    }
                }
                catch (const exception& exception)
                {
                    cout << exception.what() << endl;
                }
        }
}
```

4. Створити клас *ProductInfo*, який надає інформацію про товар. Полями класу є:

– назва товару;

– кількість.

Методи класу *ProductInfo*:

– конструктор з параметрами за замовчуванням;

– конструктор копіювання;

– гетер для поля кількості товару;

– функція покупки товару, кількість товару зменшується на 1, повертається назва товару;

– перевантаження оператора(<<);

– перевантаження оператора(=).

Створити клас *Store*. Полем класу є *vector* з типом *ProductInfo*. Методи класу *Store*:

– функція додавання товару;

– функція друку всіх наявних товарів;

– функція видалення всіх товарів, які не наявні;

– функція видалення всіх товарів у проміжку [перший індекс, другий індекс);

– перевантаження оператора квадратних дужок для доступу до елемента вектору.

Розв'язання задачі 4:

ProductInfo.h

```
#include <iostream>
```

```
using namespace std;

class ProductInfo
{
private:

	//Назва товару
	string name;

	//Кількість
	int amount;

public:

	//Конструктор з параметрами за замовчуванням
	ProductInfo(const string& name = "",
		    const int amount = 0);

	//Конструктор копіювання
	ProductInfo(const ProductInfo& product);

	//Функція для покупки товару
	string buyProduct();

	//Гетер для поля кількості товару
	int getAmount() const;

	//Перевантаження оператора(<<)
	friend ostream& operator<<(ostream& out,
				   const ProductInfo& product);

	//Перевантаження оператора(=)
	ProductInfo& operator=(const ProductInfo& product);
};
```

ProductInfo.cpp

```
#include "ProductInfo.h"

//Конструктор з параметрами за замовчуванням
ProductInfo::ProductInfo(const string& name, const int amount)
{
	this->name = name;
	this->amount = amount;
```

```
}

//Конструктор копіювання
ProductInfo::ProductInfo(const ProductInfo& product)
{
	this->name = product.name;
	this->amount = product.amount;
}

//Функція для покупки товару
string ProductInfo::buyProduct()
{
	if (amount == 0)
		throw exception("Товар відсутній!");

	this->amount--;
	return this->name;
}

//Гетер для поля кількості товару
int ProductInfo::getAmount() const
{
	return this->amount;
}

//Перевантаження оператора(=)
ProductInfo& ProductInfo::operator=(const ProductInfo& product)
{

	this->name = product.name;
	this->amount = product.amount;

	return *this;
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const ProductInfo& product)
{
	return out << "Назва: " << product.name
		<< " | Кількість: " << product.amount;
}
```

Store.h

```
#include "ProductInfo.h"
```

```
#include <vector>

//Псевдонім для ітератора
typedef vector<ProductInfo>::iterator storage_it;

class Store
{
private:

	//Контейнер vector для зберігання
	//інформації про наявні товари на складі магазину
	vector<ProductInfo> storage;

public:

	//Функція додавання товару
	void add(const string& name, const int amount);

	//Функція друку всіх наявних товарів
	void printInfo() const;

	//Функція видалення всіх товарів, які не наявні
	void removeMissingItems();

	//Функція видалення товарів у проміжку [fIndex , sIndex)
	void removeItemsInBetween(const int fIndex,
				  const int sIndex);

	//Перевантаження оператора квадратних дужок
	ProductInfo& operator[](const int index);
};
```

Store.cpp

```
#include "Store.h"

//Функція додавання товару
void Store::add(const string& name, const int amount)
{
	storage.push_back(ProductInfo(name, amount));
}

//Функція друку всіх наявних товарів
void Store::printInfo() const
{
```

```
	if (storage.empty())
		throw exception("Склад порожній!");

	for (int i = 0; i < storage.size(); i++)
	{
		cout << storage[i] << endl;
	}
}

//Функція видалення всіх товарів, які не наявні
void Store::removeMissingItems()
{
	if (storage.empty())
		throw exception("Склад порожній!");

	storage_it removeIndex;

	for (int i = 0; i < storage.size(); i++)
	{
		if (!storage[i].getAmount()) {

			removeIndex = storage.begin() + i;
			storage.erase(removeIndex);
		}
	}
}

//Функція видалення товарів у проміжку [fIndex ; sIndex)
void Store::removeItemsInBetween(const int fIndex,
				const int sIndex)
{
	if (fIndex < 0 || fIndex > storage.size())
		throw exception("Некоректний перший індекс!");

	if (sIndex < 0 || sIndex > storage.size())
		throw exception("Некоректний другий індекс!");

	if ((sIndex - fIndex) < 0)
		throw exception("Некоректний інтервал!");

	storage.erase(storage.begin() + fIndex,
			storage.begin() + sIndex);
}
```

```
//Перевантаження оператора квадратних дужок
ProductInfo& Store::operator[](const int index)
{
	if (index < 0 && index >= storage.size())
		throw exception("Некоректний індекс!");

	return storage[index];
}
```

Practice_task_4.cpp

```
#include <Windows.h>
#include "Store.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Store myStore;
    vector<string> shoppingCart;

    try
    {
        myStore.add("Конструктор", 2);
        myStore.add("Пазли", 5);
        myStore.add("Лялька", 7);

        cout << " --- Друк наявних товарів ---\n";
        myStore.printInfo();

        //Кладемо у візок всі товари під індексом 0
        shoppingCart.push_back(myStore[0].buyProduct());
        shoppingCart.push_back(myStore[0].buyProduct());

        cout << "\nПродуктовий візок: " << endl;
        for (int i = 0; i < shoppingCart.size(); i++)
        {
            cout << i + 1 << ". " << shoppingCart[i] << endl;
        }

        //Видалення товарів, які не в наявності
        myStore.removeMissingItems();

        cout << "\n--- Оновлення список ---\n";
```

```
        myStore.printInfo();

        //Видалення всіх наявних товарів
        myStore.removeItemsInBetween(0, 2);

        cout << "\n--- Друк порожнього списку товарів ---\n";
        myStore.printInfo();

    }
    catch (const exception& exception)
    {
        cout << exception.what() << endl;
    }

    return 0;
}
```

Контрольні запитання

1. Що входить до складу бібліотеки STL?
2. Які типи контейнерів наявні у STL? Наведіть кілька прикладів для кожного.
3. Як можна отримати доступ до елемента контейнера?
4. На які типи поділяються ітератори? Назвіть особливість кожного.
5. Перерахуйте основні операції над ітераторами.
6. За допомогою чого досягається швидкий доступ до елементів асоціативних контейнерів?
7. Чи є можливість змінити ключ в асоційованому контейнері?
8. Яка відмінність у додаванні елемента в асоціативному контейнері порівняно з послідовним?

10. ФУНКТОРИ

Мета: Оволодіння практичними навичками роботи з функціональними класами та об'єктами в мові C++

***Теми для попереднього опрацьовування*:**

- Перевантаження операторів.
- Шаблонні функції.
- Шаблонні класи.
- Контейнерні класи.

Теоретичні відомості

У мові програмування C++, термін "***функтор***" відноситься до об'єкта, який може використовуватися як функція, зокрема в контексті функцій вищого порядку або алгоритмів, які приймають функції як аргументи.

Основна ідея функторів полягає в тому, що об'єкт може бути викликаний як функція, коли до нього застосовується оператор виклику функції *operator*(). Це дає можливість передавати функтори як аргументи до інших функцій, зберігати їх у контейнерах, використовувати для здійснення операцій над даними та інші корисні взаємодії.

Наприклад, ось приклад функтора, який реалізує операцію додавання:

```
class AddFunctor {
public:
    int operator()(int a, int b) {
        return a + b;
    }
};
```

Цей функтор можна викликати так само, як і функцію:

```
AddFunctor add;
int result = add(3, 5);              // Результат буде 8
```

Функтор – це загальний термін, який вказує на будь-який об'єкт або структуру, яка може бути викликана як функція. Функтори можуть бути реалізовані за допомогою класів або структур. Функтори можуть зберігати стан і внутрішні дані між викликами.

Функціональний клас – це термін, який може вказувати на певний клас, що реалізує функціональність функтора. Тобто це клас, який може бути викликаний як функція, і часто він має внутрішні дані та метод *operator*().

Функціональний клас – клас, серед методів якого є перевизначений оператор виклику функції:

```
class myclass{
public:
 int operator() (int a, int b){
     return (a > b || a % 10 == 0 ? a : b);
 }
};
…
myclass A;
cout << A(4, 7);                //7
```

Функтори часто використовуються разом з алгоритмами бібліотеки STL (Standard Template Library) для обробки даних. Наприклад:

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

class SquareFunctor {
public:
    int operator()(int x) {
        return x * x;
    }
};

int main() {
    vector<int> numbers = {1, 2, 3, 4, 5};
    SquareFunctor square;

    transform(numbers.begin(), numbers.end(),
              numbers.begin(), square);
    for (int num : numbers) {
        cout << num << " ";  // Виведе: 1 4 9 16 25
    }
}
```

У цьому прикладі функтор SquareFunctor використовується разом з алгоритмом *std::transform* для піднесення кожного числа до квадрату.

Функтори надають гнучкість та можливості для створення параметризованих функцій, які можуть бути передані і використані у різних контекстах.

Використання стандартних функціональних об'єктів

greater<int>() – це функціональний об'єкт зі стандартної бібліотеки C++, який використовується для порівняння двох значень типу *int* з точки зору "*більше за*". Цей функціональний об'єкт насправді є прикладом функтору. Він реалізований як шаблонний клас, який перевантажує *operator*() для виконання порівнянь.

less<int>() – це ще один функціональний об'єкт зі стандартної бібліотеки C++, який використовується для порівняння двох значень типу int з точки зору "*менше за*". Це також є прикладом функтору, який реалізований як шаблонний клас та перевантажує *operator*() для порівнянь.

Ці функціональні об'єкти можна використовувати для порівняння значень при сортуванні, використанні алгоритмів, де потрібне порівняння, тощо.

***Приклад* 1.** Використання *greater<int>*() для порівняння двох чисел.

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

int main() {
    greater<int> greaterThan;
    int a = 5;
    int b = 3;
    if (greaterThan(a, b))
    {
        cout << a << " is greater than " << b << endl;
    } else
     {
        cout << a << " is not greater than " << b << endl;
    }
}
```

У цьому прикладі *greater<int>* використовується для порівняння двох чисел *a* та *b*, і результат порівняння виводиться на екран.

***Приклад* 2.** Використання стандартних функціональних об'єктів.

```
vector <int> v;
    for (int i = 0; i < 5; i++) {
        v.push_back(rand() % 10);
        cout << setw(4) << v[i];      // 1 7 4 0 9
    }
   sort (v.begin(), v.end(), greater<int>());
    for (const auto& x : v)
        cout << setw(4) << x;         // 9 7 4 1 0

  sort (v.begin(), v.end(), less<int>()); // за умовчанням
    for (const auto& x : v)           // 0 1 4 7 9
        cout << setw(4) << x;
```

***Приклад* 3.** Використання стандартних функціональних об'єктів. Підрахувати кількість елементів вектору, який розглянуто в прикладі 2, більше п'яти.

```
//1. Створимо предикат
 bool gr5(int a) {
    return(a >5);
}
cout << count_if(v.begin(), v.end(), gr5);          // 2

//2. Створимо функціональний об'єкт
 class _gr5 {
   public:
      bool operator()(int a){
          return (a > 5);
      }
  };
cout << count_if(v.begin(), v.end(), _gr5());       // 2

// через об'єкт класу
 _gr5 ob;
cout << count_if(v.begin(), v.end(), ob);           // 2
```

```
// 3. Створимо функціональний об'єкт : вираз замість
// константи
class _gr {
    int n;
  public:
    _gr(int _n) { n = _n; }
    bool operator() (int a) {
         return (a > n);
    }
};
cout << count_if(v.begin(), v.end(), _gr(5));        // 2

//4. Стандартні функціональні об'єкти та зв'язувачі
#include <functional>
 cout << count_if(v.begin(),v.end(),
         bind2nd(greater <int>(), 5))<<endl;         // 2

cout << count_if(v.begin(),v.end(),
         bind1st(less <int>(), 5)) << endl;          // 2
```

***Приклад* 4.** Використання функціонального об'єкту. Для словника *map* <*string*, *int*> відсортувати об'єкти по ключу по убуванню.

```
//1. Стандартний функціональний об'єкт
map <string, int, greater<string>> D2;
    D2["Johnson"] = 123;
    D2["Smith"] = 543;
    D2["Shaw"] = 999;
    D2["Atherton"] = 111;
    for (const auto& x : D2)
        cout << x.first << " " << x.second << endl;
                                // Smith 543
                                // Shaw 999
                                // Johnson 123
                                // Atherton 111

//2. Клас функціонального об'єкту compare2
class  compare2  {
    public:
       // Функція - член operator() класу compare2
```

```
        // визначає відношення більше для ключів.
         bool operator()(string s, string t) const  {
             return s > t;
         }
    };
map <string, int, compare2> D;
    D["Johnson"] = 123;
    D["Smith"] = 543;
    D["Shaw"] = 999;
    D["Atherton"] = 111;

    for (const auto& x : D)
         cout << x.first << " "<<x.second<< endl;
                                // Smith 543
                                // Shaw 999
                                // Johnson 123
                                // Atherton 111
```

***Приклад* 5.** Використання функціонального об'єкту. Для словника *map <Person, int>* реалізувати можливість різного сортування по полях класу.

```
class Person {
    string name;
    int  age;
  public:
    Person():name(""),age(0) {}
    Person(string a, int b) {
      name = a; age = b;
    }
    friend ostream& operator<<(ostream& s,
                               const Person& ob) {
        return s << ob.name << "\t" << ob.age;
    }
//функція порівняння по імені - однакове ім'я
//видаляється
friend bool operator < (const Person& d1,
                        const Person& d2){
        return d1.name < d2.name;
}
```

```
// Функтор для сортування за віком повинен бути
// константним методом! Однаковий вік видаляється
    bool operator()(const Person& d1,
                    const Person& d2) const {
         return d1.age < d2.age;
    }

    string get_name()const {
       return name;
 }
};

int main()
{
  setlocale(LC_ALL, "ru");
  string names[5] = {"Вася", "Коля", "Антон", "Саша",
"Антон"};
    int voz[5] = { 22, 22, 24, 25, 22};

  // Використання функції-порівняння
  // сортування по name
map <Person, int> m1;
int count = 0;
for (int i = 0; i < 5; i++)
        m1.insert({ Person(names[i],voz[i]),  ++count });
for (auto p = begin(m1); p != end(m1); p++)
        cout << p->first << " " << p->second << endl;
                                // Антон 24 3
                                // Вася 22 1
                                // Коля 22 2
                                // Саша 25 4
// або друк так (лише поле first (інформація з map))
for(const auto &x:m1)
        cout << x.first <<  endl;
                                // Антон 24
                                // Вася 22
                                // Коля 22
                                // Саша 25
```

```
// Використання функтора
// сортування по age
map <Person, int, Person> m2;
  count = 0;
for (int i = 0; i < 5; i++)
  m2.insert({ Person(names[i],voz[i]),++count });
for (const auto& x : m2) cout << x.first << endl;
                                // Вася 22
                                // Антон 24
                                // Саша 25
// Друк одного поля з ключа first (name або age)
// Друк лише name
for (auto p = begin(m1); p != end(m1); p++)
 cout << const_cast<Person&>(p->first).get_name() << endl;
                                // Антон
                                // Вася
                                // Коля
                                // Саша
// Друк лише age
for (auto p = begin(m1); p != end(m1); p++)
     cout << ((Person&)p->first).get_age() << endl;
// або
for (auto p = begin(m1); p != end(m1); p++)
        cout << (p->first).get_age() << endl;
                                // 24
                                // 22
                                // 22
                                // 25
// Видалити  з m1 людей, молодше за 24 роки
for (auto p = begin(m1); p != end(m1); ){
  if (const_cast<Person&>(p->first).get_age() < 24)
       p = m1.erase(p);
  else
      p++;
}
for (const auto& x : m1) // після видалення
        cout << x.first << endl;
} //main()
                                // Антон
                                // Саша
```

Задачі

Загальні умови

Кожне завдання містить приклад використання функціональних класів та функціональних об'єктів в мові С++, і може містити умови вирішення. Після формулювання завдання надано рішення у вигляді програмного коду.

1. Створити клас *City*, поля якого:

– назва міста;

– кількість населення.

Методи класу:

– конструктор з параметрами за замовчуванням;

– конструктор копіювання.

Перевантаження:

– перевантаження оператора(<<);

– перевантаження оператора(<), який порівнює назви міст;

– перевантаження оператора круглих дужок (функтор), для порівняння кількості населення.

Створити *vector* з типом *City*, заповнити його різними містами. За допомогою вбудованого *sort* спочатку звичайно відсортувати вектор, роздрукувати його. Повторно відсортувати за допомогою вбудованого функтора в клас *City*, роздрукувати.

Розв'язання задачі 1:

City.h

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

class City
{
private:

	//Назва міста
	string cityName;
```

```
        //Кількість населення
        int population;

public:

        //Конструктор з параметрами за замовчуванням
        City(const string& cityName = "",
             const int population = 0);

        //Конструктор копіювання
        City(const City& city);

        //Перевантаження оператора(<<)
        friend ostream& operator<<(ostream& out,
                                   const City& city);

        //Перевантаження оператора(<), порівнює назви міст
        friend bool operator<(const City& firstCity,
                              const City& secondCity);

        //Функтор перевантажений для порівняння
        //кількості населення
        bool operator()(const City& firstCity,
                        const City& secondCity) const;
};
```

City.cpp

```
#include "City.h"

//Конструктор з параметрами за замовчуванням
City::City(const string& cityName, const int population)
{
        this->cityName = cityName;
        this->population = population;
}

//Конструктор копіювання
City::City(const City& city)
{
        this->cityName = city.cityName;
        this->population = city.population;
}

//Функтор перевантажений для порівняння кількості населення
bool City::operator()(const City& firstCity,
```

```
                        const City& secondCity) const
{
        return firstCity.population > secondCity.population;
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const City& city)
{
        return out << "Назва: " << city.cityName
                << " | населення: " << city.population;
}

//Перевантаження оператора(<), порівнює назви міст
bool operator<(const City& firstCity, const City& secondCity)
{
        return firstCity.cityName < secondCity.cityName;
}
```

Practice_task_1.cpp

```
#include <Windows.h>
#include <algorithm>
#include <vector>
#include "City.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    vector<City> cities = {City("Харків", 1420000),
                           City("Запоріжжя", 710000),
                           City("Київ", 2890000),
                           City("Львів", 731000)};

    cout << " --- Друк інформації про міста ---\n";
    for (auto& city : cities) {
        cout << city << endl;
    }

    //Сортування vector за допомогою
    //перевантаженого operator<
    sort(cities.begin(), cities.end());

    cout << "\n--- Сортування за допомогою operator< ---\n";
```

```
    for (const auto& city : cities) {
        cout << city << endl;
    }

    //Сортування vector за допомогою функтора
    //реалізованого у класі City
    sort(cities.begin(), cities.end(), City());

    cout << "\n--- Сортування за допомогою функтора ---\n";
    for (const auto& city : cities) {
        cout << city << endl;
    }
}
```

2. Створити клас *Sphere*. Поля класу:

– колір кулі(*string*);

– об'єм кулі.

Методи класу:

– конструктор з параметром за замовчуванням;

– конструктор копіювання.

Перевантаження:

– перевантаження оператора(<), який порівнює назви кольорів;

– перевантаження оператора круглих дужок (функтор), який порівнює об'єм куль;

– перевантаження оператора(<<).

Розв'язання задачі 2:

Sphere.h

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

class Sphere
{
private:

        //Колір кулі
        string color;

        //Об'єм кулі
```

```
        double volume;

public:

        //Конструктор з параметрами за замовчуванням
        Sphere(const string& color = "",
               const double volume = 0);

        //Конструктор копіювання
        Sphere(const Sphere& sphere);

        //Перевантаження оператора(<)
        friend bool operator<(const Sphere& firstSphere,
                              const Sphere& secondSphere);

        //Функтор, порівнює об'єм куль
        bool operator()(const Sphere& firstSphere,
                        const Sphere& secondSphere) const;

        //Перевантаження оператора(<<)
        friend ostream& operator<<(ostream& out,
                                   const Sphere& sphere);
};
```

Sphere.cpp

```
#include "Sphere.h"

//Конструктор з параметрами за замовчуванням
Sphere::Sphere(const string& color, const double volume)
{
        this->color = color;
        this->volume = volume;
}

//Конструктор копіювання
Sphere::Sphere(const Sphere& sphere)
{
        this->color = sphere.color;
        this->volume = sphere.volume;
}

//Функтор, порівнює об'єм куль
bool Sphere::operator()(const Sphere& firstSphere,
                        const Sphere& secondSphere) const
```

```
{
	return firstSphere.volume > secondSphere.volume;
}

//Перевантаження оператора(<)
//порівнює назви кольорів
bool operator<(const Sphere& firstSphere,
		const Sphere& secondSphere)
{
	return firstSphere.color < secondSphere.color;
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const Sphere& sphere)
{
	return out << "Колір кулі: " << sphere.color
		<< " ,об'єм: " << sphere.volume;
}
```

Practice_task_2.cpp

```
#include <Windows.h>
#include <set>
#include "Sphere.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    const int arraySize = 4;
    Sphere arraySphere[arraySize] = {Sphere("Зелений",89.81),
                                     Sphere("Жовтий",3.94),
                                     Sphere("Срібний",17.05),
                                     Sphere("Нефритовий",486.39)
};

    //Створення контейнера set<Sphere>
    //При сортуванні використовується перевантажений operator<
    set<Sphere> sphereSetFirst;
    for (int i = 0; i < arraySize; i++)
    {
        sphereSetFirst.insert(*(arraySphere + i));
    }
```

```
    cout << " --- Друк set<Sphere> ---\n";
    for (const auto& sphere : sphereSetFirst)
        cout << sphere << endl;


    //Створення контейнера set<Sphere, Sphere>
    //При сортуванні використовується функтор,
    //який сортує за об'ємом
    set<Sphere, Sphere> sphereSetSecond;
    for (int i = 0; i < arraySize; i++)
    {
        sphereSetSecond.insert(*(arraySphere + i));
    }

    cout << "\n--- Друк set<Sphere, Sphere> ---\n";
    for (const auto& sphere : sphereSetSecond)
        cout << sphere << endl;
}
```

3. Створити клас-функтор *EvenOddFunctor*, який підраховує суму парних та непраних чисел. Поля класу:

– сума парних чисел(*int*);

– сума непарних чисел(*int*).

Методами класу є група "гетерів" для полів класу. Перевантажити оператор круглих дужок для підрахунку суми парних та непарних чисел.

Для демонстрації програми створити *vector<int>* та заповнити його. За допомогою циклу підрахувати суму парних та непарних чисел та роздрукувати.

Розв'язання задачі 3:

EvenOddFunctor.h

```
class EvenOddFunctor
{
private:

        //Сума парних чисел
        double evenSum;

        //Сума непарних чисел
        double oddSum;
```

```
public:

	//Конструктор за замовчуванням
	EvenOddFunctor();

	//Функтор, який підраховує суму парних та
	//непарних чисел
	void operator()(const int number);

	//Група "гетерів" класу
	double getEvenSum() const;
	double getOddSum() const;
};
```

EvenOddFunctor.cpp

```
#include "EvenOddFunctor.h"

//Конструктор за замовчуванням
EvenOddFunctor::EvenOddFunctor()
{
	this->evenSum = 0;
	this->oddSum = 0;
}

//Функтор, який підраховує суму парних та
//непарних чисел
void EvenOddFunctor::operator()(const int number)
{
	if (number % 2 == 0)
		evenSum += number;
	else
		oddSum += number;
}

//Група "гетерів" класу
double EvenOddFunctor::getEvenSum() const
{
	return this->evenSum;
}

double EvenOddFunctor::getOddSum() const
{
	return this->oddSum;
```

```
}
```

Practice_task_3.cpp

```
#include <Windows.h>
#include <iostream>
#include <vector>
#include "EvenOddFunctor.h"

using namespace std;

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	EvenOddFunctor functor;

	//Вектор з типом int
	vector<int> vectorInt = { 12, 31, 55, 78,
				  14, 99, 37, 0};

	//Підрахунок суми парних та
	//непарних чисел у векторі
	for (const auto& elem : vectorInt)
		functor(elem);

	cout << "Сума парних чисел vector<int>: "
		<< functor.getEvenSum() << endl;
	cout << "Сума непарних чисел vector<int>: "
		<< functor.getOddSum() << endl;
}
```

4. Створити функтор-інтерфейс *IBinaryFunction*, в якому наявне тільки абстрактне перевантаження оператора круглих дужок.

Від функтор-інтерфейсу успадковуються два класи-функтори, один перевантажений для розрахунку суми чисел(*double*), другий для розрахунку множення чисел(*double*).

Створити клас *VectorsCalculator*, полем якого є вектор з типом *double*. Методом класу є функція друку вектору. У класі перевантажити оператор

круглих дужок для арифметичних операцій над парами елементів двох векторів. Параметри:

– перший вектор;

– другий вектор;

– функтор(тип *IBinaryFunction**).

Для демонстрації програми створити та заповнити два вектори з типом *double*. Через покажчик на базовий клас динамічно виділяти пам'ять для класів-функторів. За допомогою об'єкту класу *VectorsCalculator* провести розрахунки та роздрукувати їх.

Розв'язання задачі 4:

IBinaryFunction.h

```
//Клас-функтор інтерфейс
class IBinaryFunction
{
public:

	//Абстрактне перевантаження оператора круглих дужок
	virtual double operator()(const double fNumber,
		const double sNumber) const = 0;
};
```

Add.h

```
#include "IBinaryFunction.h"

class Add : public IBinaryFunction
{
public:

	//Перевизначення перевантаження
	//оператора круглих дужок
	virtual double operator()(const double fNumber,
		const double sNumber) const override;
};
```

Add.cpp

```
#include "Add.h"
```

```
//Реалізація перевантаження
//оператора круглих дужок
double Add::operator()(const double fNumber,
	const double sNumber) const
{

	//Повертаємо суму першого та другого чисел
	return fNumber + sNumber;
}
```

Multiply.h

```
#include "IBinaryFunction.h"

class Multiply : public IBinaryFunction
{
public:

	//Перевизначення перевантаження
	//оператора круглих дужок
	virtual double operator()(const double fNumber,
		const double sNumber) const override;
};
```

Multiply.cpp

```
#include "Multiply.h"

//Реалізація перевантаження
//оператора круглих дужок
double Multiply::operator()(const double fNumber,
	const double sNumber) const
{

	//Повертаємо множення першого та другого чисел
	return fNumber * sNumber;
}
```

VectorsCalculator.h

```
#include "Add.h"
#include "Multiply.h"
#include <iostream>
#include <vector>
```

```
using namespace std;

class VectorsCalculator
{
private:

	//Результуючий вектор
	vector<double> resultVector;

public:

	//Функтор для розрахунків двох векторів
	void operator()(const vector<double>& firstVect,
			const vector<double>& secondVect,
			const IBinaryFunction* functor);

	//Функція друку результату
	void printResult() const;
};
```

VectorsCalculator.cpp

```
#include "VectorsCalculator.h"

//Функтор для розрахунків двох векторів
void VectorsCalculator::operator()(
			const vector<double>& firstVect,
			const vector<double>& secondVect,
			const IBinaryFunction* functor)
{

    //Якщо вектор не порожній,
    //очищаємо його
    if (!resultVector.empty())
        resultVector.clear();

    for (int i = 0; i < firstVect.size() && i <
secondVect.size(); i++)
    {

        //В аргументі методу push_back викликаємо функтор
        //з елементами двох векторів
        resultVector.push_back((*functor)(firstVect[i],
                                          secondVect[i]));
```

```
    }
}

//Друк результуючого вектору
void VectorsCalculator::printResult() const
{
    if (resultVector.empty())
        throw exception("Вектор порожній!");

    for (const auto& element : resultVector)
        cout << element << " ";
}
```

Practice_task_4.cpp

```
#include <Windows.h>
#include "VectorsCalculator.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    try
    {
        //Створення двох векторів
        vector<double> firstVector = { 12, 33.3, 25, 56,
                                       67, 2, 5.2, 9 };

        vector<double> secondVector = { 3, 5, 4, 10,
                                        11.5, 22.6, 66.81, 9.99 };

        //За допомогою властивостей поліморфізму
        //виділяємо динамічно пам'ять для класу-функтора
        IBinaryFunction* binaryFunc = new Add();

        VectorsCalculator vectCalculator;

        //Розрахунок суми елементів двох векторів
        vectCalculator(firstVector, secondVector, binaryFunc);

        cout << "Сума елементів векторів: ";
        vectCalculator.printResult();

        //Звільнення пам'яті
```

```
        delete binaryFunc;

        binaryFunc = new Multiply();

        //Розрахунок множення елементів двох векторів
        vectCalculator(firstVector, secondVector, binaryFunc);

        cout << "\nМноження елементів векторів: ";
        vectCalculator.printResult();

        //Звільнення пам'яті
        delete binaryFunc;
    }
    catch (const exception& exception)
    {
        cout << exception.what() << endl;
    }
}
```

5. Створити клас-функтор *Compare*, у якому наявне перевантаження оператора круглих дужок для порівняння двох чисел(*int*).

Реалізувати функцію-предикат, яка порівнює два числа з типом *int*.

Для демонстрації програми створити чотири контейнери *map* з ключем типу *int*, значенням типу *string*. Третій параметр шаблону:

– у першому *map* стандартний функтор *less*;

– у другому *map* стандартний функтор *greater*;

– у третьому *map* прототип функції-предикату, в аргумент конструктора передати ім'я створеного предикату;

– у четвертому *map* функтор *Compare*.

Створити функтор *Print*, з перевантаженням оператора круглих дужок для друку контейнера *map*. Роздрукувати результати створених контейнерів.

Розв'язання задачі 5:

Compare.h

```
#include <iostream>
#include <map>

using namespace std;
```

```
//Клас-функтор для порівняння
//двох чисел
class Compare
{
public:
	bool operator()(const int fValue, const int sValue)
const;
};

//Функція-предикат, яка порівнює два числа
bool myGreater(const int fValue, const int sValue);
```

Compare.cpp

```
#include "Compare.h"

//Функтор, який порівнює два числа
bool Compare::operator()(const int fValue, const int sValue)
const
{
	return fValue > sValue;
}

//Реалізація функції-предикату
bool myGreater(const int fValue, const int sValue)
{
	return fValue > sValue;
}
```

Print.h

```
//Клас-функтор для
//друку контейнера map
class Print
{
public:

	//У перевантаженні оператора
	//використовується шаблон
	template<typename Map>
	void operator()(const Map& map) const;

};
```

```
//Метод реалізований у файлі .h
//для того, щоб не оголошувати явно шаблони
template<typename Map>
inline void Print::operator()(const Map& map) const
{
	for (const auto& element : map) {
		cout << "В наявності: " << element.first
			<< " " << element.second << endl;
	}
}
```

Practice_task_5.cpp

```
#include <Windows.h>
#include <functional>
#include "Compare.h"
#include "Print.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Print printMap;

    //Створення першого map
    map<int, string, less<int>> firstMap = {{10,"яблук"},
        {15, "груш"},{21, "арбуз"}, {30, "слив"}};

    cout << " --- map<int, string, less<int>> ---\n";
    printMap(firstMap);

    //Створення другого map
    map<int, string, greater<int>> secondMap = { {10,"яблук"},
                {15, "груш"}, {21, "арбуз"}, {30, "слив"} };

    cout << "\n--- map<int, string, greater<int>> ---\n";
    printMap(secondMap);

    //Створення третього map
    map<int, string, bool(*)(int, int)> thirdMap(myGreater);
    thirdMap.insert(firstMap.begin(), firstMap.end());

    cout << "\n--- Сортування за допомогою myGreater() ---\n";
```

```
    printMap(thirdMap);

    //Створення четвертого map
    map<int, string, Compare> fourthMap(firstMap.begin(),
                                        firstMap.end());

    cout << "\n---Сортування за допомогою класу-функтора---\n";
    printMap(thirdMap);
}
```

Контрольні запитання

1. Перевантаження якого оператора обов'язкове для функтора?
2. Яке основне призначення функторів?
3. Перелічіть стандартні функціональні об'єкти.
4. За допомогою чого можна змінювати умову стандартних функціональних об'єктів?
5. Як можна задати тип сортування при додаванні елемента в асоціативний контейнер?
6. Які переваги використання функторів порівняно зі звичайними функціями? Які недоліки?

11. АЛГОРИТМИ STL

Мета: Оволодіння практичними навичками роботи з різними алгоритмами STL в мові C++

***Теми для попереднього опрацьовування*:**

- Функтори.
- Функціональні класи.
- Контейнерні класи.
- Алгоритми STL.

Теоретичні відомості

Щоб використовувати алгоритми бібліотеки, необхідно включити в програму заголовний файл <***algorithm***>.

Деякі алгоритми STL, призначені для обробки числових даних, визначаються в заголовному файлі <***numeric***>.

Будь-який алгоритм STL працює з одним або декількома *інтервалами*, заданими за допомогою *ітераторів*. Для першого інтервалу зазвичай задаються обидві межі (початок і кінець), а для інших інтервалів часто достатньо одного початку, тому що кінець визначається кількістю елементів у першому інтервалі.

Алгоритми працюють в режимі заміни, а не в режимі вставки, тому перед викликом алгоритму необхідно переконатися в тому, що приймальний інтервал містить достатня кількість елементів. Спеціальні ітератори вставки переводять алгоритм в режим вставки.

Для підвищення потужності і гнучкості деякі алгоритми дозволяють передавати призначені для користувача операції, які викликаються при внутрішній роботі алгоритму. Такі операції оформляються у вигляді *функцій* або об'єктів функцій.

Функція, що повертає логічне значення, називається ***предикатом***.

Функція (або об'єкт функції), що визначає *унарний* предикат, може передаватися алгоритму пошуку як критерій пошуку. Унарний предикат перевіряє, чи відповідає елемент заданому критерію (тобто до яких елементів повинна застосовуватися операція). Наприклад, це дозволяє

знайти перший елемент із значенням, меншим 50.

Функція (або об'єкт функції), що визначає *бінарний* предикат, може передаватися алгоритму сортування як критерій сортування. Бінарний предикат порівнює два елементи. Наприклад, за допомогою бінарного предиката можна відсортувати об'єкти, що представляють людей, по прізвищах і по іменах.

Предикати також використовуються для модифікації операцій в чисельних алгоритмах. Наприклад, алгоритм *accumulate*(), що зазвичай обчислює суму елементів, також дозволяє обчислювати добуток всіх елементів.

По назві алгоритму можна отримати перше уявлення про його призначення. Проєктувальники STL ввели два спеціальні суфікси.

Суфікс ***_if*** використовується за наявності двох схожих форм алгоритму з однаковою кількістю параметрів:

– в першій формі передається значення,

– а в другій – функція або об'єкт функції.

В цьому випадку версія без суфікса *_if* використовується при передачі значення, а версія з суфіксом *_ if* - при передачі функції або об'єкту функції. Наприклад,

- *find***()** шукає елемент із заданим значенням,

- *find_if***()** елемент, що задовольняє критерію, визначеному у вигляді функції або об'єкту функції.

Суфікс ***_copy*** означає, що алгоритм не лише обробляє елементи, але і копіює їх в приймальний інтервал.

Наприклад,

- *reverse*() переставляє елементи інтервалу в зворотному порядку,

- *reverse_copy*() копіює елементи в інший інтервал в зворотному порядку.

Класифікація алгоритмів (по групах)

– Немодифікуючі алгоритми

– Модифікуючі алгоритм

– Алгоритми сортування

– Чисельні алгоритми

Немодифікуючі алгоритми, які представлені в таблиці 1, зберігають як порядок дотримання оброблюваних елементів, так і їх значення. Вони працюють з ітераторами введення і прямими ітераторами і тому можуть викликатися для всіх стандартних контейнерів.

Таблиця 1 – Немодифікуючі алгоритми

Назва	*Опис*
for_each()	Виконує операцію з кожним елементом (* - можлива модифікація!)
count()	Повертає кількість елементів
count_if()	Повертає кількість елементів, що задовольняють заданому критерію
min_element()	Повертає елемент з мінімальним значенням
max_element()	Повертає елемент з максимальним значенням
find()	Шукає перший елемент із заданим значенням
find_if()	Шукає перший елемент, що задовольняє заданому критерію
search_n()	Шукає перші n послідовних елементів із заданими властивостями
search()	Шукає перше входження підінтервалу
find_end()	Шукає останнє входження підінтервалу
find_first_of()	Шукає перший з декількох можливих елементів
adjacent_find()	Шукає два суміжні елементи, рівних по заданому критерію
equal()	Перевіряє, чи рівні два інтервали
mismatch()	Повертає перший елемент, що розрізняється, в двох інтервалах
lexicographical_compare()	Перевіряє, що один інтервал менше іншого по лексикографічному критерію

***Приклад* 1.** Немодифікуючі алгоритми *count*, *count_if*. Рахують кількість значень послідовності, рівних заданому, або кількість значень, для яких справедливий заданий предикат.

```
bool pr(int x) { return (x < 15); }  //предикат
vector <int> d { 1,2,15,1,2,15,1,2,1,15, 25 };

// рахує кількість елементів, які дорівнюють 15
cout << count(d.begin(), d.end(), 15) << endl;       //3

// рахує кількість елементів, які менше за 15
cout << count_if(d.begin(), d.end(),
  bind2nd(less <long> (), 15)) << endl;              //7

// рахує кількість елементів, які більше за 15
cout << count_if(d.begin(), d.end(),
  bind2nd(greater<int>(), 15)) << endl;              //1

// рахує кількість елементів, які менше за 15
cout << count_if(d.begin(), d.end(), pr) << endl;    //7
```

***Приклад* 2.** Немодифікуючі алгоритми *find*, *find_if.* Повертають ітератор на перший елемент послідовності, рівний заданому, або для якого справедливий заданий унарний предикат (пошук).

```
bool pr(int x) { return (x < 15); }  //предикат
vector <int> d{ 1,2,15,1,2,15,1,2,1,15, 25 };

// знайти в векторі d елемент, рівний 15
cout << *(find(d.begin(), d.end(), 15));            //15

// правильний варіант використання алгоритму
 vector <int>::iterator it1;
    if ((it1 = find(d.begin(), d.end(), 15)) == d.end())
        cout << "Not found";
    else
        cout << *it1;                                //15

// вивести всі значення, менше 15
 vector <int>::iterator it2;
it1 = d.begin();
    while (it1 != d.end())
   {
        it2 = find_if(it1, d.end(), pr);
        if (it2 == d.end())
            break;
        cout << *it2 << " ";
        it1 = (++it2);               //1 2 1 2 1 2 1
    }
```

***Приклад* 3.** Немодифікуючі алгоритми *for_each*, *equal.*

```
void mod5(long x) { cout << x % 5 << " "; }
vector <int> d{ 1,2,15,1,2,15,1,2,1,15, 25 };

//for_each – викликає для кожного елемента послідовності
// задану функцію.
// Вивести залишки від ділення кожного елемента d на 5
for_each(d.begin(), d.end(), mod5);
                              //1 2 0 1 2 0 1 2 1 0 0
```

```
// equal - порівнює дві послідовності на попарний збіг
// елементів (повертає 1, якщо збігаються і 0 - якщо
// немає).
 int A1[] = { 3, 1, 4, 1, 5, 9, 3 };
 int A2[] = { 3, 1, 4, 2, 8, 5, 7, 2, 5 };
 const int N = sizeof(A1) / sizeof(int);

  cout << "Result of comparison: "
       << equal(A1, A1 + N, A2) << endl;
                                // Result of comparison: 0
  cout << "Result of comparison: "
       << equal(A1, A1 + 3, A2) << endl;
                                // Result of comparison: 1
```

Модифікуючі алгоритми, які представлені в таблиці 2, змінюють значення елементів. Модифікація виробляється безпосередньо усередині інтервалу або в процесі копіювання в інший інтервал. Якщо елементи копіюються в приймальний інтервал, вихідний інтервал залишається без змін.

Таблиця 2 – Модифікуючі алгоритми

Назва	*Опис*
for_each()	Виконує операцію з кожним елементом
copy()	Копіює інтервал, починаючи з першого елементу
copy_backwards()	Копіює інтервал, починаючи з останнього елементу
transform()	Модифікує (і копіює) елементи; об'єднує елементи двох інтервалів
merge()	Виробляє злиття двох інтервалів
swap_ranges()	Міняє місцями елементи двох інтервалів
fill()	Замінює кожен елемент заданим значенням
fill_n()	Замінює n елементів заданим значенням
generate()	Замінює кожен елемент результатом операції
generate_n()	Замінює n елементів результатом операцій
replace()	Замінює елементи із заданим значенням іншим значенням
replace_if()	Замінює елементи, відповідні критерію, заданим значенням
replace_copy()	Замінює елементи із заданим значенням при копіюванні інтервалу
replace_copy_if()	Замінює елементи, відповідні критерію, при копіюванні інтервалу

***Приклад* 4.** Модифікуючі алгоритми *copy*, *copy_backward*. Алгоритми поелементно копіюють вхідну последовательность1 у вихідну последовательность2 (яка задана лише одним ітератором). Перший алгоритм переміщає елементи у бік збільшення ітераторів вихідної послідовності, другий – у бік зменшення.

```
#include <iterator>

    char S1[] = "Hello, world!";
    char S2[] = "world";
    const int N1 = sizeof(S1) - 1;
    const int N2 = sizeof(S2) - 1;

// замінити в S1 слово "Hello" рядком S2
 copy(S2, S2 + N2, S1);
 cout << S1 << endl;
 copy(S1, S1 + N1, ostream_iterator <char>(cout));
                                        // world world
//або
 copy_backward(S2, S2+N2, S1+N2);
 copy(S1, S1 + N1, ostream_iterator <char>(cout));
                                        // world world
```

***Приклад* 5.** Модифікуючі алгоритми *copy*, *copy_backward.* Використання алгоритмів для вставки.

```
vector <int> v1 { 40,30,20,10 };
vector <int> v2 { 1,2,3 };
vector <int> v3 { 50, 60, 70 };
list <int> ll;

copy(begin(v1), end(v1), front_inserter(ll)); //в початок
        for (auto const& x : ll) cout << x << " ";
                                        // 10 20 30 40
copy(begin(v2), end(v2), back_inserter(ll));  //в кінець
        for (auto const& x : ll) cout << x << " ";
                                  // 10 20 30 40 1 2 3

auto p = ll.begin();
advance(p, 4);
copy(begin(v3), end(v3), inserter(ll, p));
                             // куди вказує покажчик p
        for (auto const& x : ll) cout << x << " ";
                         // 10 20 30 40 50 60 70 1 2 3
```

***Приклад* 6.** Модифікуючі алгоритми *fill*, *fill_n.* Ці алгоритми дозволяють заповнити послідовність (або декілька її елементів) заданим значенням.

```
vector <int> d { 1, 2, 3, 4 };
// замінити в d всі елементи на число 113
 fill(d.begin(), d.end(), 113);
    for (auto const& x : d) cout << x << " ";
                                        // 113 113 113 113
// замінити в d перші два елементи на число 113
  fill_n(d.begin(), 2, 113);
       for (auto const& x : d) cout << x << " ";
                                        // 113 113 3 4
```

***Приклад* 7.** Модифікуючі алгоритми *generate*, *generate_n.* Алгоритми замінюють елементи послідовності значеннями функції-генератора, яка вказується при виклику алгоритму.

```
// заповнити послідовність випадковими числами
// із заданого інтервалу
class my_gen {
    int a, b;
  public:
    my_gen(int a, int b) {
        srand((unsigned)time(NULL));
        this->a = a; this->b = b;
    }
    int operator() () {
        return a + (rand() % (b - a));
    }
};
//-------------------------------------
vector <int> d(10);
 generate(d.begin(), d.end(), my_gen(10, 30));
           for (auto const& x : d) cout << x << " ";
                       // 18 21 24 20 19 22 24 25 25 28
 generate_n(d.begin(), 5, my_gen(-10, -5));
           for (auto const& x : d) cout << x << " ";
                       // -7 -9 -6 -10 -6 22 24 25 25 28
```

***Приклад* 8.** Модифікуючі алгоритми *generate*, *generate_n.* Алгоритми заповнюють елементи послідовності значеннями функції-генератора, яка вказується при виклику алгоритму.

```
// заповнити послідовність випадковими числами
// із заданого інтервалу
int fun() { return (rand() % (-10 + 5) -5); }

vector <int> d(10);
 generate(d.begin(), d.end(), my_gen(10, 30));
           for (auto const& x : d) cout << x << " ";
                       // 20 17 20 23 12 11 16 12 15 19
 generate_n(d.begin(), 5, fun);
           for (auto const& x : d) cout << x << " ";
                       // -4 -5 -4 -4 -3 11 16 12 15 19
```

***Приклад* 9.** Модифікуючі алгоритми *replace*, *replace_if*, *remove*, *remove_if.*

replace, *replace_if* – ці алгоритми *замінюють* елементи послідовності із заданим старим значенням (або що задовольняють заданій умові) на нове значення.

remove, *remove_if* – ці алгоритми *видаляють* елементи послідовності із заданим значенням (або що задовольняють заданій умові), переносячи елементи, що залишилися, в початок послідовності. Місце, що звільнилося, заповнюється "сміттям". Функції повертають ітератор на початок "сміття".

```
vector <int> d{ 1, 22, 3, 44, 5, 6 };
// замінити в послідовності елементи,
// які більше 10, на 10
replace_if(d.begin(), d.end(),
        bind2nd(greater<long>(), 10), 10);
for (auto const& x : d) cout << x << " ";
                                      // 1 10 3 10 5 6
// видалити з послідовності всі елементи, які більше 5, і
// вивести в стандартний потік cout елементи, що залишилися
copy(d.begin(),
remove_if(d.begin(),d.end(),bind2nd(greater <long>(), 5)),
ostream_iterator <long> (cout, " "));         // 1 3 5
```

```
// для фізичного видалення цих елементів можна записати
// наступні оператори
bool pr1(int x) {
  return (x > 5);
}
d.erase(remove_if(d.begin(), d.end(), pr1), d.end());
 cout << d.size() << endl;                    // 3
 for (auto const& x : d) cout << x << " ";    // 1 3 5
```

***Приклад* 10.** Модифікуючі алгоритми *transform.* Алгоритм використовує операцію, яка повертає модифікований аргумент. Результат привласнюється вихідному елементу.

```
int square(int elem){
  return elem * elem;
}
vector <int> d{ 1, 2, 3, 4, 5, 6 };
transform(d.begin(), d.end(), d.begin(), square);
for (auto const& x : d) cout << x << " ";
                                        // 1 4 9 16 25 36
```

Алгоритми сортування, які представлені в таблиці 3, є окремим випадком перестановочних алгоритмів, оскільки вони теж змінюють порядок дотримання елементів. Проте сортування є складнішою операцією і зазвичай займає більше часу, чим прості перестановки. На практиці ці алгоритми вимагає підтримка ітераторів довільного доступу (для приймача).

Таблиця 3 – Алгоритми сортування

Назва	*Опис*
sort()	Сортує всі елементи
stable_sort()	Сортує із збереженням порядку дотримання рівних елементів
partial sort()	Сортує до тих пір, поки перші n елементів не будуть впорядковані правильно
nth_element()	Сортує елементи зліва і праворуч від елементу у позиції n
partition()	Змінює порядок дотримання елементів так, що елементи, відповідні критерію, виявляються спереду
stable_partition()	Те ж, що і partition(), але із збереженням відносного розташування елементів, відповідних і не відповідних критерію
make_heap()	Перетворить інтервал в купу
push_heap()	Додає елемент в купу
pop_heap()	Видаляє елемент з купи
sort_heap	Сортує купу (яка після виклику перестає бути купою)

***Приклад* 11.** Алгоритми сортування *sort*.

```
// Відсортувати вектор за збільшенням і по убуванню
class classcomp {
  public:
    bool operator()(const int& lhs, const int& rhs) const
    {
        return lhs < rhs;
    }
};
vector <int> d{ 1, -2, 3, -4, 5, -6 };
sort(d.begin(), d.end());                    //less<int>()
for (auto const& x : d) cout << x << " ";
                                          // -6 -4 -2 1 3 5
sort(d.begin(), d.end(), greater<int>());
for (auto const& x : d) cout << x << " ";
                                          // 5 3 1 -2 -4 -6
sort(d.begin(), d.end(), classcomp());
for (auto const& x : d) cout << x << " ";
                                          // -6 -4 -2 1 3 5
```

Задачі

Загальні умови

Кожне завдання містить приклад використання різних алгоритмів STL в мові С++, і може містити умови вирішення. Після формулювання завдання надано рішення у вигляді програмного коду.

1. Створити клас *Schoolboy* з полями:

– ім'я;

– оцінки(*vector* з типом *int*).

Методи класу:

– конструктор з параметром(ім'я);

– перевантаження конструктора з параметрами(ім'я, оцінки);

– функція додавання оцінки;

– функція підрахунку оцінок нижче заданої;

– функція підрахунку середньо арифметичного;

– функція заміни всіх шуканих оцінок на задану;

– функція сортування оцінок;

– функція друку інформації.

Для реалізації методів(крім функції друку інформації) не використовувати цикли та умовні оператори. Все реалізовувати за допомогою *STL*.

Розв'язання задачі 1:

Schoolboy.h

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

class Schoolboy
{
private:

	//Ім'я
	string name;

	//Оцінки
	vector<int> grades;

public:

	//Конструктор з параметрами
	Schoolboy(const string& name);

	//Перевантаження конструктора з параметрами
	Schoolboy(const string& name,
		  const vector<int>& grades);

	//Функція додавання оцінки
	void addGrade(const int grade);

	//Функція підрахунку оцінок нижче заданої
	int gradesCountLessGiven(const int grade) const;

	//Функція підрахунку середньо арифметичного
	double averageGrade() const;
```

```
        //Функція заміни всіх шуканих оцінок на задану
        void replaceGrades(const int searchGrade,
                           const int replaceable);

        //Функція друку інформації
        void printInformation() const;

        //Функція сортування оцінок
        void sortGrades();
};
```

Schoolboy.cpp

```
#include "Schoolboy.h"

//Конструктор з параметрами
Schoolboy::Schoolboy(const string& name)
{
        this->name = name;
}

//Перевантаження конструктора з параметрами
Schoolboy::Schoolboy(const string& name,
                     const vector<int>& grades)
{
        this->name = name;

        //Копіюємо вміст вектору
        copy(grades.begin(), grades.end(),
                back_inserter(this->grades));
}

//Функція підрахунку оцінок нижче заданої
int Schoolboy::gradesCountLessGiven(const int grade) const
{

        int count = count_if(grades.begin(), grades.end(),

                //За допомогою bind2nd зв'язуємо перший
                //аргумент функтора less<int> з grade
                bind2nd(less<int>(),
                        (grade > 13 || grade < 2) ? 6 : grade));

        return count;
```

```
}

//Функція підрахунку середньо арифметичного
double Schoolboy::averageGrade() const
{

	//За допомогою accumulate підраховуємо суму елементів
	return static_cast<double>(accumulate(grades.begin(),
				grades.end(), 0)) / grades.size();
}

//Функція заміни всіх шуканих оцінок на задану
void Schoolboy::replaceGrades(const int searchGrade, const int
replaceable)
{
	replace(grades.begin(), grades.end(),
		searchGrade, replaceable);
}

//Функція друку інформації
void Schoolboy::printInformation() const
{
	cout << "Ім'я: " << name <<
		"\nОцінки: ";

	for (const auto& element : grades)
		cout << element << " ";

	cout << endl;
}

//Функція сортування оцінок
void Schoolboy::sortGrades()
{
	sort(grades.begin(), grades.end(), less<int>());
}

//Функція додавання оцінки
void Schoolboy::addGrade(const int grade)
{
	grades.push_back((grade > 12 || grade < 1) ? 11
							: grade);
}
```

```
#include <Windows.h>
#include "Schoolboy.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    vector<int> grades = { 5, 12, 5, 10 };

    Schoolboy schoolboy("Денис", grades);

    cout << " --- Друк інформації ---\n";
    schoolboy.printInformation();
    cout << "Середнє арифметичне: "
        << schoolboy.averageGrade() << endl;
    cout << "Кількість оцінок нижче 7: "
        << schoolboy.gradesCountLessGiven(7) << endl;

    //Додавання оцінок
    schoolboy.addGrade(10);
    schoolboy.addGrade(8);
    schoolboy.addGrade(8);

    //Сортування оцінок
    schoolboy.sortGrades();

    //Змінюємо всі оцінки 5 на 8
    schoolboy.replaceGrades(5, 8);

    cout << "\n--- Друк оновленої інформації ---\n";
    schoolboy.printInformation();
    cout << "Середнє арифметичне: "
        << schoolboy.averageGrade() << endl;
    cout << "Кількість оцінок нижче 10: "
        << schoolboy.gradesCountLessGiven(10) << endl;
}
```

2. Створити клас *NumbersSet*, з полями класу:

– назва множини;

– множина чисел, використовувати *list*(числа можуть повторюватися) з типом даних *int*.

Методи класу:

– конструктор з параметром(назва);

– функція додавання числа(після додавання відразу сортувати список);

– функція зміни всіх чисел на їх квадрати(функція *transform*);

– функція об'єднання множин;

– функція видалення повторюваних чисел(функція *unique*);

– функція знаходження перетину множин(функція *set_intersection*);

– функція знаходження першої пари чисел, що розрізняються;

– функція друку множини.

Додатково реалізувати функцію розрахунку квадрату числа. Для реалізації методів класу не використовувати цикли і умовні оператори(крім функції друку).

Розв'язання задачі 2:

NumbersSet.h

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

//Псевдонім для типу даних pair<iter, iter>
typedef pair<list<int>::const_iterator,
       list<int>::const_iterator> mismatch_pair;

class NumbersSet
{
private:

       //Ім'я множини
       string name;

       //Множина чисел
       list<int> numbersSet;
```

```
public:

	//Конструктор з параметром
	NumbersSet(const string& name);

	//Функція додавання числа
	void addNumber(const int number);

	//Функція зміни всіх чисел на їх квадрати
	void changeToSquareNumber();

	//Функція об'єднання множин
	void mergeSet(NumbersSet& set);

	//Функція видалення повторюваних чисел
	void removeRepetitions();

	//Функція знаходження перетину множин
	list<int> getIntersection(const NumbersSet& set) const;

	//Функція знаходження першої пари чисел, що
	//розрізняється
	mismatch_pair firstMismatch
			(const NumbersSet& set) const;

	//Функція друку множини
	void print() const;
};

//Функція розрахунку квадрата числа
int square(const int number);
```

NumbersSet.cpp

```
#include "NumbersSet.h"

//Функція розрахунку квадрата числа
int square(const int number)
{
	return number * number;
}

//Конструктор з параметром
NumbersSet::NumbersSet(const string& name)
{
```

```
	this->name = name;
}

//Функція додавання числа
void NumbersSet::addNumber(const int number)
{
	numbersSet.push_back(number);

	//Після додавання сортуємо list
	numbersSet.sort();
}

//Функція зміни всіх чисел на їх квадрати
void NumbersSet::changeToSquareNumber()
{
	if (numbersSet.empty())
		throw exception("Множина чисел порожня!");

	transform(numbersSet.begin(), numbersSet.end(),
		numbersSet.begin(), square);
}

//Функція об'єднання множин
void NumbersSet::mergeSet(NumbersSet& set)
{
	numbersSet.merge(set.numbersSet);
}

//Функція видалення повторюваних чисел
void NumbersSet::removeRepetitions()
{
	if (numbersSet.empty())
		throw exception("Множина чисел порожня!");

	//За допомогою функції unique видаляємо всі числа,
	//які повторюються повертаючи ітератор
	//на нову послідовність
	auto it = unique(numbersSet.begin(), numbersSet.end());

	//За допомогою методу resize і функції distance
	//змінюємо фактичний розмір контейнера list
	numbersSet.resize(distance(numbersSet.begin(), it));
}

//Функція знаходження перетину множин
```

```
list<int> NumbersSet::getIntersection(const NumbersSet& set)
const
{
	if (numbersSet.empty())
		throw exception("Перша множина чисел порожня!");

	if (set.numbersSet.empty())
		throw exception("Друга множина чисел порожня!");

	list<int> intersection;

	//Пошук усіх повторів чисел у двох множинах
	set_intersection(numbersSet.begin(), numbersSet.end(),
		set.numbersSet.begin(), set.numbersSet.end(),

		//back_inserter потрібен для додавання
		//знайденого числа у кінець intersection
		back_inserter(intersection));

	return intersection;
}

//Функція знаходження першої пари чисел, що розрізняється
mismatch_pair NumbersSet::firstMismatch
				(const NumbersSet& set) const
{
	if (numbersSet.empty())
		throw exception("Перша множина чисел порожня!");

	if (set.numbersSet.empty())
		throw exception("Друга множина чисел порожня!");

	mismatch_pair pair = mismatch(numbersSet.begin(),
					numbersSet.end(),
					set.numbersSet.begin(),
					set.numbersSet.end());

	return pair;
}

//Функція друку множини
void NumbersSet::print() const
{
	if (numbersSet.empty())
		throw exception("Множина чисел порожня!");
```

```
        cout << name << " = { ";

        for (const auto& elem : numbersSet)
               cout << elem << " ";

        cout << "}" << endl;
}
```

Practice_task_2.cpp

```
#include <Windows.h>
#include "NumbersSet.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    NumbersSet firstSet("A");
    NumbersSet secondSet("B");

    try
    {

        //Заповнення множин
        //з сортуванням
        firstSet.addNumber(7);
        firstSet.addNumber(7);
        firstSet.addNumber(4);
        firstSet.addNumber(9);
        firstSet.addNumber(9);

        secondSet.addNumber(5);
        secondSet.addNumber(6);
        secondSet.addNumber(4);

        cout << " --- Друк множин ---\n";
        firstSet.print();
        secondSet.print();

        //Пошук перетину множин
        list<int> intersection;
        intersection = firstSet.getIntersection(secondSet);
```

```
        cout << "\n--- Перетин множин ---\n"
            << "I = { ";
        for (const auto& elem : intersection)
            cout << elem << " ";
        cout << "}" << endl;

        //Пошук першої пари, що не збігається
        mismatch_pair mismatchPair = firstSet.firstMismatch
                                                    (secondSet);

        cout << "\nПара чисел: " << *mismatchPair.first << ", "
            << *mismatchPair.second << endl;

        //Всі числа зводимо у квадрат
        secondSet.changeToSquareNumber();

        //Видаляємо всі повтори чисел
        firstSet.removeRepetitions();

        cout << "\n--- Друк оновлених множин ---\n";
        firstSet.print();
        secondSet.print();

        //Об'єднання множин
        firstSet.mergeSet(secondSet);

        cout << "\n--- Друк множин після об'єднання ---\n";
        firstSet.print();
        secondSet.print();
    }
    catch (const exception& exception)
    {
        cout << exception.what() << endl;
    }
}
```

3. Створити клас *Bus* з полями:

– номер автобуса;

– прапорець місцезнаходження, де 1 - скоро прибуде на зупинку, 0 - виїхав із зупинки.

Методи класу:

– конструктор з параметрами за замовчуванням;

– конструктор копіювання;

– група "гетерів" класу.

Перевантаження:

– оператора(<), який порівнює номери;

– оператор(==), який порівнює номери;

– оператор(=).

Створити класу *BusStation*, поля класу:

– назва станції;

– два вектори з типом *Bus*, один вектор зберігає інформацію про автобуси, що їдуть зі станції, другий про автобуси що приїжджають на станцію.

Методи класу:

– конструктор з параметром(назва станції);

– функція додавання автобуса;

– функція сортування списків автобусів;

– функція порівняння автобусів, що їдуть з однієї станції, з автобусами, що приїжджають на іншу станцію(функція *equal*);

– функція пошуку схожих номерів автобусів, що їдуть з першої станції, приїжджають на другу станцію;

– функція друку інформації.

Розв'язання задачі 3:

Bus.h

```
class Bus
{
private:

	//Номер автобуса
	int busNumber;

	// Прапорець місцезнаходження
	//1 - скоро прибуде на зупинку
	//0 - виїхав із зупинки
	bool locationFlag;
```

```
public:

	//Конструктор з параметрами за замовчуванням
	Bus(const int busNumber = 0,
		const bool locationFlag = 0);

	//Конструктор копіювання
	Bus(const Bus& bus);

	//Група "гетерів" класу
	int getBusNumber() const;
	bool getLocationFlag() const;

	//Перевантаження оператора(=)
	Bus& operator=(const Bus& bus);

	//Перевантаження оператора(<), для порівняння номерів
	bool operator<(const Bus& bus) const;

	//Перевантаження оператора(==), для порівняння номерів
	bool operator==(const Bus& bus) const;
};
```

Bus.cpp

```
#include "Bus.h"

//Конструктор з параметрами за замовчуванням
Bus::Bus(const int busNumber, const bool locationFlag)
{
	this->busNumber = busNumber;
	this->locationFlag = locationFlag;
}

//Конструктор копіювання
Bus::Bus(const Bus& bus)
{
	this->busNumber = bus.busNumber;
	this->locationFlag = bus.locationFlag;
}

//Група "гетерів" класу
int Bus::getBusNumber() const
{
	return this->busNumber;
```

```
}

bool Bus::getLocationFlag() const
{
	return this->locationFlag;
}

//Перевантаження оператора(=)
Bus& Bus::operator=(const Bus& bus)
{
	this->busNumber = bus.busNumber;
	this->locationFlag = bus.locationFlag;

	return *this;
}

//Перевантаження оператора(<), для порівняння номерів
bool Bus::operator<(const Bus& bus) const
{
	return busNumber < bus.busNumber;
}

//Перевантаження оператора(==), для порівняння номерів
bool Bus::operator==(const Bus& bus) const
{
	return busNumber == bus.busNumber;
}
```

BusStation.h

```
#include <iostream>
#include <algorithm>
#include <vector>
#include "Bus.h"
using namespace std;

class BusStation
{
private:

	//Назва станції
	string name;

	//Автобуси, що їдуть
	vector<Bus> departedBuses;
```

```
        //Автобуси, що приїжджають
        vector<Bus> arrivingBuses;

public:

        //Конструктор з параметром
        BusStation(const string& name);

        //Функція додавання автобуса
        void addBus(const Bus& bus);

        //Функція сортування списків автобусів
        void sortList();

        //Функція порівняння автобусів, що
        //їдуть з однієї станції
        //з автобусами, що приїжджають на іншу станцію
        bool equalDepartedArriving(BusStation& station);

        //Функція пошуку схожих номерів
        //автобусів, що їдуть/приїжджають
        vector<int> findDepartedArrivingBuses
                        (const BusStation& station);

        //Функція друку інформації
        void printInformation() const;
};
```

BusStation.cpp

```
#include "BusStation.h"

//Конструктор з параметром
BusStation::BusStation(const string& name)
{
        this->name = name;
}

//Функція додавання автобуса
void BusStation::addBus(const Bus& bus)
{
        //Якщо автобус виїхав із зупинки
        if (bus.getLocationFlag() == 0) {
```

```
                vector<Bus>::iterator it;

                //Перевірка на наявність номера у
                //списку автобусів, що приїжджають
                if ((it = find(arrivingBuses.begin(),
                                arrivingBuses.end(),
                                bus.getBusNumber()))
                        != arrivingBuses.end())
                {
                        //Якщо наявний видаляємо
                        arrivingBuses.erase(it);
                }

                departedBuses.push_back(bus);
        }
        else {
                arrivingBuses.push_back(bus);
        }
}

//Функція сортування списків автобусів
void BusStation::sortList()
{
        sort(arrivingBuses.begin(), arrivingBuses.end());
        sort(departedBuses.begin(), departedBuses.end());
}

//Функція порівняння автобусів, що їдуть з однієї станції
//з автобусами, що приїжджають на іншу станцію
bool BusStation::equalDepartedArriving(BusStation& station)
{

        if (departedBuses.size()
                != station.arrivingBuses.size())
                throw exception("Кількість автобусів, що
                приїжджають та їдуть різна!");

        //Сортуємо списки автобусів
        this->sortList();
        station.sortList();

        //Для порівняння використовуємо перевантаження
        //оператора(==) у класі Bus
        bool flag = equal(this->departedBuses.begin(),
```

```
                        this->departedBuses.end(),
                        station.arrivingBuses.begin());

        return flag;
}

//Функція пошуку схожих номерів автобусів, що їдуть/приїжджають
vector<int> BusStation::findDepartedArrivingBuses
                                    (const BusStation& station)
{
        vector<int> busNumbers;

        for (const auto& elem : this->departedBuses) {

                if (find(station.arrivingBuses.begin(),
                         station.arrivingBuses.end(), elem)
                         != station.arrivingBuses.end())
                {
                        //Якщо знайшли номер, додаємо
                        //до busNumbers
                        busNumbers.push_back(elem.getBusNumber());
                }
        }

        return busNumbers;
}

//Функція друку інформації
void BusStation::printInformation() const
{
        cout << "Назва станції: " << name;

        cout << "\nАвтобуси, що приїжджають: ";
        for (const auto& elem : arrivingBuses)
                cout << elem.getBusNumber() << " ";

        cout << "\nАвтобуси, що їдуть: ";
        for (const auto& elem : departedBuses)
                cout << elem.getBusNumber() << " ";

        cout << endl;
}
```

```
#include <Windows.h>
#include "BusStation.h"

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	BusStation firstStation("Зупинка А");
	BusStation secondStation("Зупинка Б");

	try
	{

		//Додавання автобусів до першої зупинки
		firstStation.addBus(Bus(89, 1));
		firstStation.addBus(Bus(80, 1));
		firstStation.addBus(Bus(41, 0));
		firstStation.addBus(Bus(51, 0));

		//Додавання автобусів до другої зупинки
		secondStation.addBus(Bus(99, 0));
		secondStation.addBus(Bus(41, 1));
		secondStation.addBus(Bus(11, 1));

		cout << "--- Друк інформації про зупинки ---\n";
		firstStation.printInformation();
		cout << endl;
		secondStation.printInformation();

		//Сортування списків
		firstStation.sortList();
		secondStation.sortList();

		if (firstStation.equalDepartedArriving
					(secondStation))
		{
			cout << "\nНомери автобусів збігаються"
				<< endl;
		}
		else {
			cout << "\nНомери автобусів не збігаються"
				<< endl;
		}
```

```
            cout << "\n--- Друк оновленої "
                 << "інформації про зупинки ---\n";
            firstStation.printInformation();
            cout << endl;
            secondStation.printInformation();

            vector<int> busNumbers = firstStation.
                   findDepartedArrivingBuses(secondStation);

            cout << "\nПерелік номерів, які збіглися: ";
            for (const auto& elem : busNumbers)
                   cout << elem << " ";

            cout << endl;
      }
      catch (const exception& exception)
      {
            cout << exception.what() << endl;
      }
}
```

4. Створити клас *Point*. Полями є координати точки. Методи класу:

– конструктор з параметрами за замовчуванням;

– конструктор копіювання.

Перевантаження:

– оператор(<<);

– оператор(<), який порівнює суму координат.

Створити клас-функтор *Generator*, який генерує точку з випадковими координатами. Полями класу є:

– ліва межа;

– права межа.

Перевантажити оператор круглих дужок для генерації випадкової точки. Для демонстрації програми створити *vector* з типом *Point* на 5 елементів. За допомогою функції *generate* і класу *Generator* заповнити вектор, та роздрукувати. Відсортувати вектор за допомогою *sort*. Знайти елементи з максимальною/мінімальною сумою координат.

Розв'язання задачі 4:

Point.h

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

class Point
{
private:

	//Координати точки
	int x;
	int y;

public:

	//Конструктор з параметрами за замовчуванням
	Point(const int x = 0, const int y = 0);

	//Конструктор копіювання
	Point(const Point& point);

	//Перевантаження оператора(<<)
	friend ostream& operator<<(ostream& out,
				const Point& point);

	//Перевантаження оператора(<), який порівнює
	//суму координат
	bool operator<(const Point& point) const;
};
```

Point.cpp

```
#include "Point.h"

//Конструктор з параметрами за замовчуванням
Point::Point(const int x, const int y)
{
	this->x = x;
	this->y = y;
}

//Конструктор копіювання
Point::Point(const Point& point)
{
```

```
	this->x = point.x;
	this->y = point.y;
}

//Перевантаження оператора(<), який порівнює суму координат
bool Point::operator<(const Point& point) const
{
	return (this->x + this->y) < (point.x + point.y);
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const Point& point)
{
	return out << " X = " << point.x
		   << ", Y = " << point.y;
}
```

Generator.h

```
#include "Point.h"

class Generator
{
private:

	//Ліва межа
	int leftValue;

	//Права межа
	int rightValue;

public:

	//Конструктор з параметром
	Generator(const int lValue, const int rValue);

	//Перевантаження оператора круглих
	//дужок для генерації точок
	Point operator()();
};
```

Generator.cpp

```
#include "Generator.h"
```

```
//Конструктор з параметром
Generator::Generator(const int lValue, const int rValue)
{
	srand(time(NULL));

	this->leftValue = lValue;
	this->rightValue = rValue;
}

//Перевантаження оператора круглих
//дужок для генерації точок
Point Generator::operator()()
{
	int x = leftValue + (rand() % (rightValue - leftValue));
	int y = leftValue + (rand() % (rightValue - leftValue));

	return Point(x,y);
}
```

Practice_task_4.cpp

```
#include <Windows.h>
#include <vector>
#include <algorithm>
#include "Point.h"
#include "Generator.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    //Оголошення вектору з типом Point на 5 елементів
    vector<Point> points(5);

    //Заповнюємо вектор за допомогою класу Generator
    generate(points.begin(), points.end(), Generator(1, 10));

    cout << " --- Друк масиву точок ---\n";
    for (const auto& elem : points)
        cout << elem << endl;

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

```
    cout << "\n--- Друк відсортованого масиву ---\n";
    for (const auto& elem : points)
        cout << elem << endl;

    auto iterator = max_element(points.begin(), points.end());
    cout << "\nТочка з максимальною сумою x,y: "
         << *iterator << endl;

    iterator = min_element(points.begin(), points.end());
    cout << "Точка з мінімальною сумою x,y: "
         << *iterator << endl;
}
```

Контрольні запитання

1. У якому режимі працюють алгоритми *STL*?
2. Що таке предикат? Які типи предикатів існують?
3. Для чого використовують предикати в алгоритмах *STL*?
4. Які існують типи алгоритмів *STL*?
5. Чи можна використовувати алгоритм *for_each* для модифікації контейнера? Якщо так, то як?
6. Наведіть приклади модифікуючих алгоритмів.
7. За допомогою якого алгоритму можна замінити/заповнити контейнер випадковими числами?
8. Що відбувається з елементами контейнеру після їх видалення за допомогою алгоритму *remove*?
9. Які існують алгоритми сортування *STL*?
10. Як контролювати тип сортування контейнерів у *STL*?

12. ЛЯМБДА-ВИРАЗИ

Мета: Оволодіння практичними навичками роботи з лямбда-виразами в мові С++

***Теми для попереднього опрацьовування*:**

- Функтори.
- Функціональні класи.
- Контейнерні класи.
- Алгоритми STL.

Теоретичні відомості

Лямбда-вираз (або просто "*лямбда*") в програмуванні дозволяє визначити анонімну функцію усередині іншої функції. Лямбда-вирази в С++ – це коротка форма запису анонімних функторів.

```
class classcomp
{
  public:
    bool operator()(const int& lhs, const int& rhs) const
    {
        return lhs < rhs;
    }
};
vector <int> d{ 1, -2, 3, -4, 5, -6 };
sort(d.begin(), d.end(), classcomp());
    for (auto const& x : d) cout << x << " ";

sort(d.begin(), d.end(),
     [](const int& lhs, const int& rhs)
              {return lhs < rhs; }
     );
for (auto const& x : d) cout << x << " ";
```

В лямбді немає типу, якого можна було б використовувати явно. Компілятор генерує унікальний тип лямбди, який нам не видно.

Лямбда складається з *трьох частин* :

[ ] – список захвату;

() – список параметрів;
{} – тіло.

```
[](){};        // визначаємо порожній лямбда-вираз
```

Список захвату [ ]

Змінні усередині [ ] доступні усередині лямбди у вигляді копії або посилання. Захоплені змінні стають частиною лямбди і не передаються при виклику.

```
int a = 10;
// auto f = []() {return a * 9; } ; //ERROR
```

Поле [ ] використовується для того, щоб надати (побічно) лямбді доступ до змінних з навколишньої зони видимості, до яких вона зазвичай не має доступу.

```
auto f = [a]() {return a * 9; }; //передача за значенням
cout << f();                                          //90

//виклик може бути таким
cout << [a]() {return a * 9; }() << endl;             //90
```

Коли виконується лямбда-визначення, то для кожної захоплюваної змінної усередині лямбди створюється копія цієї змінної (з ідентичним ім'ям). Дані змінні-копії ініціалізуються за допомогою змінних із зовнішньої зони видимості з тим же ім'ям.

Можна захоплювати змінні *за посиланням*, щоб дозволити лямбді впливати на значення аргументів.

```
auto f1 = [&a]() {return ++a; }; //передача за посиланням
cout << f1();                                        //11
cout << "a = " << a << endl;                         // a = 11

auto b = f1();// Виклик лямбди-функції. Змінна a береться
              // із списку захвату і тут не передається
cout << b << "  " << a << endl;                      // 12 12

int x = 5;
```

```
auto f = [x]() {cout << x; };
    f();                                                    //5
```

Можна захопити *декілька змінних*, розділивши їх комами (може бути як захват за значенням, так і захват за посиланням).

```
int a = 10, b = 10;
auto f = [&a, b]() {return a++ +  b; };
cout << f() << endl;                                        // 20
cout << " a = " << a << " b = " << b << endl;
                                                  // a = 11 b = 10
```

Захвати за умовчанням – це можливість автоматичної генерації списку змінних, які потрібно захопити, тобто всі змінні, згадані в лямбді :

= для захвату за значенням;

& для захвату за посиланням.

```
int a = 1, b = 2, c = 3;
auto f = [=]() {return a + b + c; };
cout << f() << endl;                                        // 6
cout <<" a = "<< a <<"  b = "<< b <<"  c = "<< c;
                                      // a = 1  b = 2  c = 3

//--------------------------------------
int a = 1, b = 2, c = 3;
auto f = [&]() {return a++ + b++ + c++; };
cout << f() << endl;                                        // 6
cout <<" a = "<< a <<"  b = "<< b <<"  c = "<< c;
                                      // a = 2  b = 3  c = 4
```

Захвати за умовчанням можуть бути змішані із звичайними захватами. Сповна допускається захопити деякі змінні за значенням, а інші – за посиланням, але при цьому кожна змінна може бути захоплена лише один раз.

```
// Захват змінних a і b за значенням, а c – за посиланням
[a, b, &c](){};
// Захват змінної c за посиланням, а всіх інших – за
// значенням
[=, &c](){};
// Захват змінної a за значенням, а всіх інших – за
```

```
// посиланням
[&, a](){};
// Заборонено, оскільки вже визначили захват за посиланням
// для всіх змінних
[&, &armor](){};
// Заборонено, оскільки вже визначили захват за значенням
// для всіх змінних
[=, armor](){};
// Заборонено, оскільки змінна a використовується двічі
[a, &b, &a](){};
// Заборонено, оскільки захват за умовчанням має бути
// першим елементом в списку захвату
[a, &](){};
```

Список параметрів ()

Такий же, як і звичайних функцій.

```
   int m = 5;
   auto f = [m](int a) {return a * m; };           // a = 2

// тут стався захват m = 5!!
   cout << f(2);                                    // 10

// виклик може бути таким
    cout << [m](int a) {return a * m; }(a);         // 10

     m = 10;
     int a = 2;
     cout << f(a);                                  // 10

// Повторне використання f із зміненою зовнішньою
// змінною
int m = 5;
auto f = [&m](int a) {return a * m; };
cout << f(2);                                       // 10
m = 10;
cout << f(2);                                       // 20
```

Тип повернення **(ТП)**

За замовчуванням виводиться тип лямбда-виразу, що повертається.

```
[](){ return true; };                          // ТП bool
[]()-> bool{ return true };           // явна вказівка ТП
```

```
int m = 5;
auto f = [&m](int a) -> bool {return a * m; };
cout << f(2) << endl;                                  // 1
cout << f(0) << endl;                                  // 0
```

Якщо в лямбді-функції використовувався оператор *return*, то все останні ТП усередині лямбди-функції мають бути такими ж.

```
auto f = [](int x, int y, bool b) {
        if (b) return x / y;
        else return static_cast<double>(x) / y;
    };            // Тут тип double конфліктує
                  // з типом int, що було виведено раніше
```

Якщо в лямбді-функції використовувався різні ТПЗ, то треба:

1) виконати явні перетворення до одного типу;

2) або явно вказати ТП для лямбди-функції.

```
auto f = [](int x, int y, bool b)-> double
{
   if (b)
      return x / y;
   else
     return static_cast<double>(x) / y;
};
cout << f(3,2,true) << endl;                          // 1
cout << f(3, 2, false);                               // 1.5

// виклик може бути таким
cout << [](int x, int y, bool b)-> double {
        if (b) return x / y;
        else return static_cast<double>(x) / y;
} (3,2,true);
```

***Приклад* 1.** Дани дві змінні $i = 3, j = 5$. Написати лямбду-функцію, яка

підсумовує ці змінні.

```
int i = 3, j = 5;
auto f = [i, j]()
{
  return i + j;
};
cout << f() << endl;                                    //8

// виклик може бути таким
        cout << [i, j]() { return i + j;}();          // 8
```

В даному випадку лямбда-функція захоплює *i* за значенням, а *- j* за посиланням. Отже зміна *i* не відбувається.

```
int i = 3, j = 5;
auto f = [i, &j]()
{
   return i + j;
};
cout << f() << endl;                                    //8
i = 22, j = 44;
cout << f() << endl;                                    //47
```

***Приклад* 2.** Зробити лямбда-функцію, яка виводить значення-параметр і рахує, скільки разів була викликана лямбда-функція.

```
auto print = [](auto val)
{
   int count {0};
   cout << count++ << "   " << val << endl;
};

print("hello");                          // 0 hello
print("world");                          // 0 world
print(1);                                // 0 1
print(2);                                // 0 2
print("dog");                            // 0 dog
```

Якщо потрібно, щоб змінна *count* була загальною для всіх лямбд, її

потрібно оголосити *поза* лямда-функцією і передати за посиланням.

```
int count = 0;
auto print{
        [&count](auto val) {
         cout << count++ << "    " << val << endl; }
    };

print("hello");                         // 0 hello
print("world");                         // 1 world
print(1);                               // 2 1
print(2);                               // 3 2
print("dog");                           // 4 dog

// виклик може бути таким
[&count](auto val) {
        cout << count++ << "    " << val << endl;
    }("hello");
```

***Приклад* 3.** Зробити лямбда-функцію, яка виводить значення-параметр і рахує, скільки разів була викликана лямбда-функція *для кожного типу*.

```
auto print =
       [](auto val) {
           static int count {0};
           cout << count++ << "   " << val << endl; };
   print("hello");                      // 0 hello
   print("world");                      // 1 world
   print(1);                            // 0 1
   print(2);                            // 1 2
   print("dog");                        // 2 dog
```

Для кожного типу згенерувалася своя лямбда. *static* змінна не є загальною для різних типів (для кожного типу має свій рахунок).

Використання лямбди-функцій в алгоритмах

```
// Відсортувати елементи вектору
vector <int> v {1, 3, 2, 0, -7, 4};
    sort(v.begin(), v.end(),
            [](const auto& a, const auto& b) {
                        return a < b;}
    );
for (const auto& x : v)cout << setw(4) << x;
                                          // -7 0 1 2 3 4

// Знайти максимальний елемент
auto best = max_element(v.begin(), v.end(),
        [](const auto& a, const auto& b) {
            return a < b;}
    );                                  // повертає ітератор

cout << *best << endl;                                  // 4

// У векторі знайти перше парне число
vector <int> v { 1, 3, 21, 92, -7, 4 };
vector <int> ::iterator rezult =
        find_if(v.begin(), v.end(),
            [](int n) {return n % 2 == 0; }
        );
if (rezult != v.end())
        cout << *rezult << endl;                        //92

// У векторі знайти кількість непарних елементів
cout << count_if(v.begin(), v.end(),
              [](int n){ return n % 2 != 0; }
        );                                               //4

// Друк вектора за допомогою лямбди-функції
 vector <int> v { 1, -3, 2 };
    for_each(v.begin(), v.end(),
        [](int x) { cout << setw(4) << x; }
    );                                          // 1 3 -2


// Змінити вектор:
```

```
// - якщо елемент < 5, то помножити його 10;
// - якщо елемент парний, то розділити його 2;
// - для останніх - звести в квадрат.
 vector <int> v{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    transform (v.begin(), v.end(), v.begin(),
        [](int x) -> double
        {
            if (x < 5) return x * 10;
            else
                if (x % 2 == 0) return x / 2.0;
                else return x * x;
        }
    );
 ostream_iterator<double> it(cout, " ");           //друк
    copy(v.begin(), v.end(), it);
                            // 0 10 20 30 40 25 3 49 4 81

// Вивести кількість елементів вектора, що потрапляють в
// заданий інтервал
 vector <int> v { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
 int k = 2, m = 8;
 cout << count_if(
           v.begin(), v.end(),
           [k, m](const int x)
           {
              return x > k && x < m;
           }
         );                                          //5

// Знайти у векторі значення площі, розраховане по
// введених значеннях висоти і ширини
  vector <int> areas { 100, 25, 121, 40, 56 };
  int width{};
  int height{};
  cout << "Enter width and height: ";
  cin >> width >> height;


// захват за умовчанням за значенням змінних width і
```

```
// height
 auto found
{
    find_if(areas.begin(), areas.end(),
                    [=](int knownArea)
                    {
                   return (width * height == knownArea);
                    }
           )
};

 if (found == areas.end()) {
      cout << "No( \n";
 }
 else {
      cout << "Yes)\n";
}
         // Enter width and height: 10 20
         // No(

         // Enter width and height: 10 10
         // Yes)
```

Специфікатор mutable

Використовується для того, щоб змінити захоплені [ ] змінні усередині лямбди-функції, але не міняти локальну змінну.

***Приклад* 4.** Використання специфікатора *mutable*.

```
     int a = 10;
     cout << a << endl;                                  //10

     auto f = [&a]() {return ++a; };
     cout << f() << endl;                                //11
     cout << a;                                          //11

 //---------------------
     int a = 10;
     cout << a << endl;                                  //10
    auto f = [a]() mutable {return ++a; };
```

```
    cout << f() << endl;                                    //11
    cout << a;                                              //10
```

***Приклад* 5.** Використання специфікатора *mutable*.

```
vector <int> v;
int init = 0;                           // локальна змінна
generate_n(back_inserter(v),10,
        [init]() {return init++; }
);

//Error, оскільки параметр, який передається по копії,
//не може бути змінений в лямбді-виразі

//---------------------
generate_n(back_inserter(v),10,
        [init]() mutable {return init++; }
);
for_each(v.begin(), v.end(),
        [](int x) { cout << setw(4) << x; }
);
                                    // 0 1 2 3 4 5 6 7 8 9
 cout << endl << init;                                      //0
                       // не помінялася усередині лямбди
//---------------------
generate_n(back_inserter(v),10,
        [&init]() {return init++; }
    );
cout << init;                                               //10
                                  // передача за посиланням
```

***Приклад* 6.** Використання лямбда-виразів.

```
// Виклик  лямбди-функції
int n = [](int x, int y) { return x + y; }(5, 4);
    cout << n << endl;                                      // 9

// Або таким чином

 int x = 5, y = 4;
```

```
 auto n = [](int x, int y)
 {
     return x + y;
};
    cout << n(x,y) << endl;

// Вкладені лямбда-функції. Внутрішня лямбда множить
// аргумент на 2. Зовнішня до результату додає 3.

int f2 = [](int x)
  {
        return [](int x)
         {
            return x * 2;
          }                                          //5*2
        (x)+3;                                       // 10+3
   }  (5);

// Print the result.
cout << f2 << endl;                                   //13
```

Задачі

Загальні умови

Кожне завдання містить приклад використання лямбда-виразамів в мові С++, і може містити умови вирішення. Після формулювання завдання надано рішення у вигляді програмного коду.

1. Створити клас *Number*, полем класу є число з типом *int*. Методами класу є функції доступу до поля класу.

Створити *vector* з типом *Number* на 10 елементів. За допомогою функції *generate* заповнити вектор. Відсортувати вектор щоб зліва були парні числа, праворуч непарні. Відсортований вектор надрукувати за допомогою *for_each*. Підрахувати кількість непарних чисел. У функціях використовувати лямбда-вирази.

Розв'язання задачі 1:

Number.h

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

class Number
{
private:

	int value;

public:

	//Конструктор з параметром за замовчуванням
	Number(const int value = 0);

	//Конструктор копіювання
	Number(const Number& number);

	//Гетер поля класу
	int getNumber() const;

	//Сетер поля класу
	void setNumber(const int value);
};
```

Number.cpp

```
#include "Number.h"

//Конструктор з параметром за замовчуванням
Number::Number(const int value)
{
    this->value = value;
}

//Конструктор копіювання
Number::Number(const Number& number)
{
    this->value = number.value;
}

//Гетер поля класу
int Number::getNumber() const
```

```
{
    return this->value;
}

void Number::setNumber(const int value)
{
    this->value = value;
}
```

Practice_task_1.cpp

```
#include <Windows.h>
#include <vector>
#include <algorithm>
#include "Number.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    srand(time(NULL));

    vector<Number> numbers(10);
    int min = 0;
    int max = 0;

    cout << "Введіть (min, max): ";
    cin >> min >> max;

    //Генерація масиву об'єктів
    generate(numbers.begin(), numbers.end(),

        //Захоплюємо min, max за значенням.
        //Тип значення, що повертається Number
        [min,max]() -> Number{

            return Number(min + (rand() % (max - min)));
        });

    //Сортування вектор, зліва парні числа, праворуч
    //непарні числа
    sort(numbers.begin(), numbers.end(),
        [](const Number& fNumber,
           const Number& sNumber) -> bool{
```

```
            if (sNumber.getNumber() % 2 == 1 &&
                fNumber.getNumber() % 2 == 0) {
                return true;
            }
            else {
                return false;
            }
        });


    cout << "Відсортований вектор: ";
    for_each(numbers.begin(), numbers.end(),
        [](const Number& number) {

            cout << number.getNumber() << " ";
        });

    cout << "\nКількість непарних елементів: " <<
        count_if(numbers.begin(), numbers.end(),
            [](const Number& number) -> bool {
                return number.getNumber() % 2 != 0;
            });

    return 0;
}
```

---

2. Створити клас *MathRectangle*, з полями:

– ширина;

– висота.

Методи класу:

– конструктор з параметрами за замовчуванням;

– гетери полів класу;

– функція підрахунку площі.

Перевантажити оператор(>>) для ініціалізації полів.

Створити *vector* з типом *int*, який містить різні площі. За допомогою функції *find_if* з лямбда-виразом перевірити наявність площі у векторі. Створити лямбда-вираз, який друкує інформацію об'єкту класу *MathRectangle* і рахує кількість викликів.

Розв'язання задачі 2:

MathRectangle.h

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

class MathRectangle
{
private:

	//Ширина
	int width;

	//Висота
	int height;

public:

	//Конструктор з параметрами за замовчуванням
	MathRectangle(const int width = 0,
		const int height = 0);

	//Гетери для полів класу
	int getWidth() const;
	int getHeight() const;

	//Функція підрахунку площі
	int getArea() const;

	//Перевантаження оператора(>>)
	friend istream& operator>>(istream& in,
		MathRectangle& rectangle);
};
```

MathRectangle.cpp

```
#include "MathRectangle.h"

//Конструктор з параметрами за замовчуванням
MathRectangle::MathRectangle(const int width, const int height)
{
	this->width = (width > 0 ? width : 1);
	this->height = (height > 0 ? height : 1);
}
```

```
//Гетери для полів класу
int MathRectangle::getWidth() const
{
	return this->width;
}

int MathRectangle::getHeight() const
{
	return this->height;
}

//Функція підрахунку площі
int MathRectangle::getArea() const
{
	return width * height;
}

//Перевантаження оператора(>>)
istream& operator>>(istream& in, MathRectangle& rectangle)
{
	return in >> rectangle.width >> rectangle.height;
}
```

Practice_task_2.cpp

```
#include <Windows.h>
#include <vector>
#include <algorithm>
#include "MathRectangle.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    //Вектор, який містить площі
    vector<int> areas = {100, 121, 10, 20, 60};

    MathRectangle rectangle;

    cout << "Введіть (довжину, висоту): ";
    cin >> rectangle;

    //Змінна містить функцію find_if з лямбда-виразом
    auto searchArea = find_if(areas.begin(), areas.end(),
```

```
        //Лямбда-захоплення містить всі раніше оголошені
        //змінні, які передаються за допомогою посилання
        [&](const int area) {
            return area == rectangle.getArea();
        });

    //Змінна містить лямбда-вираз, який друкує
    //інформацію про прямокутник
    auto print = [](const MathRectangle& rectangle) {

        //Статична змінна, яка рахує кількість викликів
        static int counter{ 0 };

        cout << counter++ << ". Довжина = "
             << rectangle.getWidth()
            << ", висота = " << rectangle.getHeight()
            << ", площа = " << rectangle.getArea() << endl;
    };

    //Перший виклик print
    print(rectangle);

    cout << "\nВведіть (довжину, висоту): ";
    cin >> rectangle;

    //Другий виклик print
    print(rectangle);

    if (searchArea != areas.end()) {
        cout << "\nПлощу знайдено!" << endl;
    }
    else {
        cout << "\nПлощу не знайдено!" << endl;
    }
}
```

---

3. Створити клас-функтор *ArrayFile*, в якому наявне перевантаження оператора круглих дужок. Перевантажений оператор повертає випадкове число з файлу.

Оголосити *vector* з типом *int* на 20 елементів. Заповнити вектор таким чином:

– індекс вектору менше 5, кожне наступне число збільшується на 1;

– індекс у межах від 5 до 10, кожне число збільшується на 5;

– індекс більше 10, кожне наступне число збільшується вдвічі.

Роздрукувати вектор. Змінити всі числа у векторі за допомогою функції *transform*. Якщо число менше 1000 помножити його на 3, в іншому випадку зчитати випадкове число з файлу і замінити.

Розв'язання задачі 3:

ArrayFile.h

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

//Клас-функтор для роботи з файлом
class ArrayFile
{
public:

	//Перевантажений оператора круглих дужок ,який
	//повертає випадкове число з масиву
	int operator()(const string& path)const;
};
```

ArrayFile.cpp

```
#include "ArrayFile.h"

int ArrayFile::operator()(const string& path) const
{

	//Допоміжний вектор
	vector<int> tempVector;

	ifstream fin(path);

	if (!fin.is_open())
		throw exception("Файл не відкрився!");

	int number = 0;
	while (fin >> number)
```

```
        {
                tempVector.push_back(number);
        }

        fin.close();

        if (tempVector.size() == 0)
                throw exception("Вектор порожній!");

        //Повертаємо випадкове значення вектору
        return tempVector[rand() % tempVector.size()];
}
```

Practice_task_3.cpp

```
#include <Windows.h>
#include <algorithm>
#include "ArrayFile.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    srand(time(NULL));

    //Оголошуємо вектор на 20 елементів
    vector<int> array(20);

    //Оголошення об'єкту класу ArrayFile
    ArrayFile randomNumberFile;

    //Шлях до файлу з масивом
    string path = "arrayFile.txt";

    //Змінна, яка відстежує індекс вектору
    int index = 0;

    //Змінна, яка відповідає з якого числа
    //починається послідовність
    int initValue = 0;

    generate(array.begin(), array.end(),
        [index, initValue]() mutable {

            int returnedValue = 0;
```

```
        if (index < 5) {

            returnedValue = ++initValue;
        }
        else if (index >= 5 && index < 10) {

            returnedValue = initValue += 5;
        }
        else {

            returnedValue = initValue *= 2;
        }

        //Збільшуємо індекс, але
        //зовні змінна не збільшується
        index++;
        return returnedValue;
    });

cout << " --- Заповнений вектор ---\n";
for_each(array.begin(), array.end(),
    [](const int value) {
        cout << value << " ";
    });

//Зміна вектору
transform(array.begin(), array.end(), array.begin(),

    //randomNumberFile передаємо за допомогою посилання,
    //path передаємо за значенням(конструктор копіювання)
    [&randomNumberFile, path](const int value) {

        //Якщо значення більше 1000, повертаємо
        //випадкове число з файлу
        if (value > 1000){
            return randomNumberFile(path);
        }
        else {
            return value * 3;
        }
    });

cout << "\n\n--- Змінений вектор ---\n";
for_each(array.begin(), array.end(),
```

```
        [](const int value) {
            cout << value << " ";
        });
}
```

4. Створити клас *Animal* з полями:

– назва тварини;

– вага.

Методи класу:

– конструктор з параметрами за замовчуванням;

– конструктор копіювання;

– група гетерів класу.

У класі наявне перевантаження оператора(<<).

Створити клас *Zoo* з полями:

– змінна, яка контролює тип додавання і сортування(*flag*);

– список тварин(*list*).

Методи класу:

– конструктор за замовчуванням;

– сетер для поля *flag*;

– функція додавання тварини, якщо *flag* дорівнює 0, додаємо на початок списку, в іншому випадку додаємо в кінець списку;

– функція пошуку тварини, реалізувати за допомогою функції *for_each* та лямбда-виразу;

– сортування списку тварин, якщо *flag* дорівнює 1, сортуємо за назвою, в іншому випадку за вагою.

Розв'язання задачі 4:

Animal.h

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

class Animal
{
private:

        //Назва тварини
```

```
        string name;

        //Вага
        int weight;
public:

        //Конструктор з параметрами за замовчуванням
        Animal(const string& name = "",
                const int weight = 0);

        //Конструктор копіювання
        Animal(const Animal& animal);

        //Група гетерів класу
        string getName() const;
        int getWeight() const;

        //Перевантаження оператора(<<)
        friend ostream& operator<<(ostream& out,
                                   const Animal& animal);
};
```

Animal.cpp

```
#include "Animal.h"

//Конструктор з параметрами за замовчуванням
Animal::Animal(const string& name, const int weight)
{
        this->name = name;
        this->weight = weight;
}

//Конструктор копіювання
Animal::Animal(const Animal& animal)
{
        this->name = animal.name;
        this->weight = animal.weight;
}

//Група гетерів класу
string Animal::getName() const
{
        return this->name;
}
```

```
int Animal::getWeight() const
{
	return this->weight;
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const Animal& animal)
{
	return out << "Назва тварини: " << animal.name
		   << ", вага: " << animal.weight;
}
```

Zoo.h

```
#include "Animal.h"
#include <list>
#include <algorithm>

class Zoo
{
private:
	//Змінна, яка контролює
	//тип додавання і сортування
	bool flag;

	//Список тварин
	list<Animal> animals;

public:
	//Конструктор за замовчуванням
	Zoo();

	//Сетер для поля flag
	void setFlag(const bool flag);

	//Функція додавання тварини
	void addAnimal(const Animal& animal);

	//Функція пошуку тварини
	bool findAnimal(const string& animal) const;

	//Функція сортування списку
	void sortList();
```

```
        //Функція друку інформації
        void printInfo() const;
};
```

Zoo.cpp

```
#include "Zoo.h"

//Конструктор за замовчуванням
Zoo::Zoo()
{
        this->flag = 0;
}

//Сетер для поля flag
void Zoo::setFlag(const bool flag)
{
        this->flag = flag;
}

//Функція додавання тварини
void Zoo::addAnimal(const Animal& animal)
{
        if (this->flag)
                animals.push_back(animal);
        else
                animals.push_front(animal);
}

//Функція пошуку тварини
bool Zoo::findAnimal(const string& searchedAnimal) const
{

        bool findFlag = 0;

        for_each(animals.begin(), animals.end(),
                [&](const Animal& animal) {

                        //Якщо знайшли тварину, змінюємо findFlag
                        if (searchedAnimal == animal.getName()) {
                                findFlag = 1;
                        }
                });

        return findFlag;}
```

```
//Функція сортування списку
void Zoo::sortList()
{
	animals.sort(

		// Щоб отримати доступ до змінних класу
		// у лямбда-виразі
		// потрібно передати покажчик this
		// в іншому випадку до полів/методів
		//класу неможливо звернутися
		[this](const Animal& fAnimal,
			const Animal& sAnimal) -> bool {

			if (this->flag)
				return fAnimal.getName() <
					sAnimal.getName();
			else
				return fAnimal.getWeight() >
					sAnimal.getWeight();
		});
}
//Функція друку інформації
void Zoo::printInfo() const
{
	for (const auto& animal : this->animals)
		cout << animal << endl;
}
```

Practice_task_4.cpp

```
#include <Windows.h>
#include "Zoo.h"

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	Zoo zoo;

	//Зміна прапорця
	zoo.setFlag(1);

	//Додавання тварин у кінець списку
```

```
        zoo.addAnimal(Animal("Тигр", 150));
        zoo.addAnimal(Animal("Лев", 190));
        zoo.addAnimal(Animal("Зебра", 280));
        zoo.addAnimal(Animal("Слон", 3000));

        cout << "--- Друк інформації ---\n";
        zoo.printInfo();

        //Зміна прапорця
        zoo.setFlag(0);

        //Сортування за вагою тварини
        zoo.sortList();

        cout << "\n\n--- Відсортований список ---\n";
        zoo.printInfo();

        if (zoo.findAnimal("Зебра")) {
                cout << "\nТварина знайдена!\n";
        }
        else {
                cout << "\nТварина не знайдена!\n";
        }
}
```

5. Створити клас *RandomFunction*. У класі наявне перевантаження оператора круглих дужок, яке повертає випадкову математичну функцію(лямбда-вираз). Наявні функції:

– функція підсумовування;

– функція віднімання;

– функція множення;

– функція ділення.

Також у класі наявна функція, яка перевіряє правильність введеного знаку.

За допомогою класу створити міні-гру, у якій треба відгадати знак математичного виразу. Для цього треба:

– згенерувати два числа;

– згенерувати випадкову математичну функцію, і підрахувати результат виразу;

– роздрукувати вираз на екрані, він має такий вид: "x ? y = z";

– зчитати з екрана знак і перевірити його.

Розв'язання задачі 5:

RandomFunction.h

```
#include <iostream>
#include <functional>

using namespace std;

class RandomFunction
{
public:

	//Перевантаження оператора круглих дужок, яке
	//повертає випадкову математичну функцію.
	//std::function<> використовується, для того, щоб
	//вказати компілятору, що повертається функція з
	//відповідною сигнатурою. Сигнатура функції, що
	//повертається вказується у трикутних дужках
	function<int(const int, const int)> operator()();

	//Функція, яка перевіряє правильність введеного знаку
	bool symbolCheck(const int fValue, const int sValue,
		const int result,const char symbol) const;
};

//Функція, яка генерує випадкове число у межах [min, max]
int getRandomNumber(const int min, const int max);
```

RandomFunction.cpp

```
#include "RandomFunction.h"

//Функція, яка генерує випадкове число у межах [min, max]
int getRandomNumber(const int min, const int max)
{
	return min + rand() % (max - min + 1);
}

//Перевантаження оператора круглих дужок, яке
//повертає випадкову математичну функцію.
function<int(const int, const int)> RandomFunction::operator()()
```

```
{       //Генеруємо випадкове число
        switch (getRandomNumber(1,4))
        {
        case 1:
                //Повертаємо лямбда-вираз, функція підсумовування
                return [](const int fValue, const int sValue)
                {return fValue + sValue; };

                break;
        case 2:
                //Повертаємо лямбда-вираз, функція віднімання
                return [](const int fValue, const int sValue)
                {return fValue - sValue; };

                break;
        case 3:
                //Повертаємо лямбда-вираз, функція множення
                return [](const int fValue, const int sValue)
                {return fValue * sValue; };

                break;
        case 4:
                //Повертаємо лямбда-вираз, функція ділення
                return [](const int fValue, const int sValue)
                {return fValue / sValue; };

                break;
        default:
                break;
        }
}

//Функція, яка перевіряє правильність математичного виразу
bool RandomFunction::symbolCheck(const int fValue,
                        const int sValue, const int result,
                        const char symbol) const
{
        if (symbol != '+' && symbol != '-'
                && symbol != '*' && symbol != '/')
                throw exception("Некоректний знак!");

        //Перевірка правильності виразу
        switch (symbol)
        {
        case '+':
```

```
            return (fValue + sValue) == result;
            break;
        case '-':
            return (fValue - sValue) == result;
            break;
        case '*':
            return (fValue * sValue) == result;
            break;
        case '/':
            return (fValue / sValue) == result;
            break;
        default:
            return 0;
            break;
        }
}
```

Practice_task_5.cpp

```
#include <Windows.h>
#include "RandomFunction.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    srand(time(NULL));

    //Оголошення об'єкту класу RandomFunction
    RandomFunction mathFunctions;

    //Допоміжні змінні
    int firstNumber = 0;
    int secondNumber = 0;
    int result = 0;
    char mathSymbol;

    try
    {
        while (true)
        {

            //Генеруємо два числа
            firstNumber = getRandomNumber(10, 20);
            secondNumber = getRandomNumber(1, 10);
```

```
            //Генерація випадкової математичної функції
            auto resultFunc = mathFunctions();

            //Підраховуємо результат виразу
            result = resultFunc(firstNumber, secondNumber);

            cout << firstNumber << " ? " << secondNumber
                << " = " << result;

            cout<<"\nВведіть математичний символ(+, -, *, /): ";
            cin >> mathSymbol;

            //Перевірка введеного символу
            if (mathFunctions.symbolCheck
                (firstNumber, secondNumber, result, mathSymbol))
            {
                cout << "Ви вгадали символ!\n\n";
            }
            else {
                cout << "Ви не вгадали символ!\n\n";
            }
        }
    }
    catch (const exception& exception)
    {
        cout << exception.what() << endl;
    }
}
```

Контрольні запитання

1. Що таке лямбда-вираз? Де його можна оголошувати?

2. З яких частин складається лямбда-вираз?

3. Якими способами можна захоплювати значення в лямбда-виразі? Їх можна комбінувати?

4. Як можна захопити всі значення в лямбда-виразі?

5. Як явно вказати тип значення, що повертається, у лямбда-виразі?

6. Чи для кожного типу даних *static* змінна в лямбді є загальною?

7. Для чого у лямбда-виразах використовується специфікатор *mutable*?

8. Чи можлива вкладеність лямбда-виразу?

13. ТЕСТУВАННЯ

Мета: Оволодіння практичними навичками роботи з тестування програм і функцій за допомогою конструкції *assert*() в мові С++

***Теми для попереднього опрацьовування*:**

- Функції в мовах С/С++.
- Тестування за допомогою *assert*().

Теоретичні відомості

Assert — це спеціальна конструкція, що дозволяє перевіряти припущення про значення довільних даних в довільному місці програми. Ця конструкція може автоматично сигналізувати при виявленні некоректних даних, що зазвичай приводить до *аварійного завершення* програми з вказівкою місця виявлення некоректних даних.

Assert'и дозволяють відловлювати помилки в програмах на етапі компіляції або під час виконання. Перевірки на етапі компіляції не так важливі — в більшості випадків їх можна замінити аналогічними перевірками під час виконання програми. Тому надалі під *assert'ами* матимемо на увазі лише перевірки *під час виконання програми*.

assert ( ) – "*оператор перевірочного затвердження"* в мові С++ – це макрос препроцесора, який обробляє умовний вираз під час виконання. Якщо умовний вираз істинний, то стейтмент *assert* нічого не робить. Якщо він помилковий, то виводиться повідомлення про помилку, і програма завершується. Це повідомлення про помилку містить помилковий умовний вираз, а також ім'я файлу з кодом та номером рядка з *assert*.

Assert'и можна розділити на *наступні класи.*

1. Перевірка вхідних аргументів на початку функції.
2. Перевірка даних при виході з функції.
3. Перевірка даних, з якими працює функція, усередині коду функції.

***Приклад* 1.** Перевірка вхідних аргументів на початку функції. Функція рахує факторіал числа n. Число n повинне лежати в межах від 0 до 10 включно.

```
int factorial(int n){
          // Факторіал негативного числа не рахується
  assert(n >= 0);

/* Якщо n перевищить 10, то це може привести або до
цілочисельного переповнювання результату, або до
переповнювання стеку.*/
  assert(n <= 10);
    int result = 1;
    for (int i = 2; i <= n; ++i) {
        result *= i;
    }
  return result;
}

#include <assert.h>
int main()
{
   cout << factorial(-5);
```

```
Assertion failed: n >= 0, file D:
```

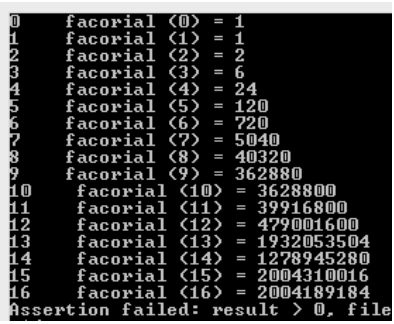
```
for (int i = 0; i < 100; ++i)
       cout << i <<"  facorial (" << i << ") = "
                << factorial(i)<<endl;
}
```

```
0    facorial (0) = 1
1    facorial (1) = 1
2    facorial (2) = 2
3    facorial (3) = 6
4    facorial (4) = 24
5    facorial (5) = 120
6    facorial (6) = 720
7    facorial (7) = 5040
8    facorial (8) = 40320
9    facorial (9) = 362880
10    facorial (10) = 3628800
Assertion failed: n <= 10, file D:
```

***Приклад* 2.** Перевірка даних при виході з функції.

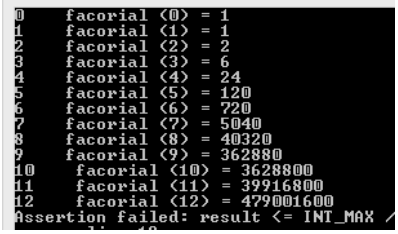
```
int factorial(int n){
  int result = 1;
  for (int i = 2; i <= n; ++i) {
         result *= i;   }
```

```
/* Як тільки n перевищить допустиму межу, станеться
цілочисельне переповнювання. В цьому випадку a[i] може
набути негативного або нульового значення */
  assert(result > 0);
  return result;
}
```



***Приклад* 3.** Перевірка даних, з якими працює функція, усередині коду функції.

```
int factorial(int n){
  int result = 1;
  while (n > 1) {
/* Ця перевірка краща, ніж перевірка з попереднього пункту
(перед виходом з функції), оскільки вона спрацьовує перед
першим переповнюванням result, тоді як перевірка з
попереднього пункту може пропустити випадок, коли в
результаті переповнювання (або серії переповнювань)
підсумкове значення result залишається позитивним.*/
    assert(result <= INT_MAX / n);
    result *= n;
    --n;
  }
  return result;
}
```



***Приклад* 4.** Необхідно зробити наступні функції:

– функція суми двох чисел, кожне > 0;

– функція сортування масиву в порядку переданому як аргументи (за зростанням/спаданням);

– тести цих двох функцій.

```
// Функція здобуття суми двох чисел >0
int sum (int  a, int b) {
   if (a>0 || b>0) {                  // помилка, треба &&
  return a + b;
    }
   else return -100;
}
// Тести для функції здобуття суми
void testSUM() {
    assert(sum(5, 4) == 9);
    assert(sum(-5, 4) == -100);
    assert(sum(5, -4) == -100);
    assert(sum(-5, -4) == -100);}

int main(){
    cout << sum(-5, 4) << endl;     //-1 Але це помилка!!!
    testSUM();
}
// Функція сортування масиву в порядку переданому як
// аргументи ((за зростанням/спаданням)
void sortArray(int data[], int N, char c) {
 for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
 {
     if (c == '>')                        // за спаданням
        if (data[i] > data[j])
             swap(data[i], data[j]);
     if (c == '<')                       // за зростанням
        if (data[i] > data[j])         // помилка, треба <
            swap(data[i], data[j]);
 }
}
```

```
// Тести для функції сортування
void testSort() {
 // початкові дані
  int data[] = {7,9,5,6,2,3,4,8,1};
 // очікуваний відсортований масив за зростанням
  int expectedData1[] = {1,2,3,4,5,6,7,8,9};
 // очікуваний відсортований масив за спаданням
  int expectedData2[] = {9,8,7,6,5,4,3,2,1 };

 sortArray(data, 9, '>');     // за спаданням
    for (int i = 0; i < 9; i++) {
        cout << data[i] << " ";
        assert(data[i] == expectedData2[i]);
    }
cout << endl;
    sortArray(data, 9, '<');     // за зростанням
    for (int i = 0; i < 9; i++) {
        cout << data[i] << " ";
        assert(data[i] == expectedData1[i]); }
}
int main()
{
    testSort();
}
```

Задачі

Загальні умови

Кожне завдання містить приклад використання конструкції *assert*() для тестування програм і функцій в мові С++ , і може містити умови вирішення. Після формулювання завдання надано рішення у вигляді програмного коду.

1. Створити клас-функтор *Sum*, який рахує суму позитивних чисел, якщо наявне хоча б одне від'ємне число, повертаємо -1. Зробити функцію з групою тестів. Тести проводити за допомогою макросу *assert*.

Створити клас-функтор *Sort*, який сортує масив. Перевантажений оператор круглих дужок зробити шаблоном. Параметрами перевантаженого оператору є:

– покажчик на масив;

– розмір масиву;

– предикат.

Створити функцію з тестом для масиву. У функції оголосити три масиви:

– випадково заповнений;

– відсортований за зростанням;

– відсортований за спаданням.

Відсортувати масив за зростанням/спаданням і порівняти з початковими масивами(*assert*).

Розв'язання задачі 1:

Sum.h

```
#include <assert.h>

#define ERROR_NEGATIVE_NUMB -1

//Клас-функтор
class Sum
{
public:

	//Перевантаження оператора круглих дужок
	int operator()(const int fValue,
		const int sValue) const;
};

//Функція з групою тестів
void testingSum();
```

Sum.cpp

```
#include "Sum.h"

//Перевантаження оператора круглих дужок
int Sum::operator()(const int fValue, const int sValue) const
```

```
{

	//Якщо числа невід'ємні
	//повертаємо суму
	if (fValue > 0 && sValue > 0) {
		return fValue + sValue;
	}

	//В іншому випадку помилку
	else {
		return -1;
	}
}

//Функція з групою тестів
void testingSum()
{
	Sum sum;

	assert(sum(10, 10) != ERROR_NEGATIVE_NUMB);
	assert(sum(99, 1) != ERROR_NEGATIVE_NUMB);
	assert(sum(-5, 7) != ERROR_NEGATIVE_NUMB);
	assert(sum(-19, -9) != ERROR_NEGATIVE_NUMB);
}
```

SortArray.h

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

//Прототип функції з групою тестів
void testingSort();

//Прототип функції друку масиву
void printArray(const int* array,const int size);

//Клас-функтор
class SortArray {

public:

	//Шаблонне перевантаження оператора круглих дужок
	template<typename Pred>
	void operator()(int* array, const int size,
```

```
                Pred pred) const;
};

/*
        Реалізація перевантаження оператора круглих дужок
        Параметрами є:
        - покажчик на масив;
        - розмір;
        - предикат.
*/
template<typename Pred>
void SortArray::operator()
        (int* array, const int size, Pred pred) const
{

        int tempNumber = 0;

        for (int i = 0; i < size - 1; i++)
        {
                for (int j = i; j < size; j++)
                {
                        if (pred(array[i], array[j])) {

                                tempNumber = *(array + i);
                                *(array + i) = *(array + j);
                                *(array + j) = tempNumber;
                        }
                }
        }
}
```

SortArray.cpp

```
#include "SortArray.h"

//Функція з групою тестів сортування
void testingSort()
{
        SortArray sort;

        //Вхідні дані
        int data[] = { 8,1,2,6,7,4,3,5,9 };

        //Очікуваний відсортовані дані за зростанням
        int sortedData[] = { 1,2,3,4,5,6,7,8,9 };
```

```
        //Очікуваний відсортовані дані за спаданням
        int reverseSortedData[] = {9,8,7,6,5,4,3,2,1};

        //Розмір масиву
        int size = sizeof(data) / sizeof(int);

        //Сортування за зростанням
        sort(data, size, std::greater<int>());
        printArray(data, size);

        //Перевірка відсортованих даних
        for (int i = 0; i < size; i++) {
                assert(data[i] == sortedData[i]);
        }

        //Сортування за спаданням
        sort(data, size, std::less<int>());
        printArray(data, size);

        //Перевірка відсортованих даних
        for (int i = 0; i < size; i++) {
                assert(data[i] == reverseSortedData[i]);
        }
}

//Функція друку масиву
void printArray(const int* array, const int size)
{
        for (int i = 0; i < size; i++)
        {
                std::cout << *(array + i) << " ";
        }
        std::cout << '\n';
}
```

Practice_task_1.cpp

```
#include <Windows.h>
#include "SortArray.h"
#include "Sum.h"

int main()
{
    SetConsoleCP(1251);
```

```
    SetConsoleOutputCP(1251);

    testingSort();
    testingSum();
}
```

2. Створити клас *ArrayFile* з полем шлях до файлу. Реалізувати методи:

– конструктор з параметром;

– сетер для поля класу;

– функція зчитування цілочисельного масиву, яка повертає *vector<int>*;

– функція заміни масиву у файлі.

Створити функцію з тестами для вектору:

– розмір вектору більше 0;

– кожен елемент більший за 0;

– сума всіх елементів менша за 1000.

Для демонстрації створити вектор, зчитати масив з файлу. Провести тест, якщо тест успішний помножити всі непарні елементи на 10 та перезаписати масив у файл.

Розв'язання задачі 2:

ArrayFile.h

```
#include <iostream>
#include <fstream>
#include <vector>
#include <assert.h>
using namespace std;

//Функція тестування вектору
void testingVector(const vector<int> vectorInt);

class ArrayFile
{
private:

        //Шлях до файлу
        string path;
```

```
public:

	//Конструктор з параметром
	ArrayFile(const string& path);

	//Сетер для поля
	void setPath(const string& path);

	//Функція зчитування масиву з файлу
	vector<int> getArray() const;

	//Функція заміни масиву у файлі
	void setArray(const vector<int> vectorInt) const;
};
```

ArrayFile.cpp

```
#include "ArrayFile.h"

//Функція тестування вектору
void testingVector(const vector<int> vectorInt) {

	int sum = 0;

	assert(vectorInt.size() != 0);

	for (int i = 0; i < vectorInt.size(); i++)
	{
		assert(vectorInt[i] >= 0);
		sum += vectorInt[i];
	}

	assert(sum < 1000);
}

//Конструктор з параметром
ArrayFile::ArrayFile(const string& path)
{
	this->path = path;
}

//Сетер для поля
void ArrayFile::setPath(const string& path)
{
```

```
	this->path = path;
}

//Функція зчитування масиву з файлу
vector<int> ArrayFile::getArray() const
{
	vector<int> vectorInt;

	ifstream fin(path);

	//Перевірка на коректність
	//відкриття файлу
	assert(fin);

	int tempNumber = 0;
	while (fin >> tempNumber)
	{
		vectorInt.push_back(tempNumber);
	}

	fin.close();
	return vectorInt;
}

//Функція заміни масиву в файлі
void ArrayFile::setArray(const vector<int> vectorInt) const
{
	ofstream fout(path);

	//Перевірка на коректність
	//відкриття файлу
	assert(fout);

	for (int index = 0; index < vectorInt.size(); index++)
	{
		fout << vectorInt[index] << " ";
	}

	fout.close();
}
```

Practice_task_2.cpp

```
#include <Windows.h>
#include <algorithm>
```

```
#include <iomanip>
#include "ArrayFile.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    const string path = "array.txt";
    ArrayFile file(path);

    vector<int> vectorInt = file.getArray();

    cout << " --- Початковий масив ---\n";
    for_each(vectorInt.begin(), vectorInt.end(),
        [](const int number) { cout << setw(4) << number; });
    cout << '\n';

    testingVector(vectorInt);

    for_each(vectorInt.begin(), vectorInt.end(),
        [](int& number) { number % 2 == 1 ? number *= 10
                                          : number; });

    cout << "\n--- Змінений масив ---\n";
    for_each(vectorInt.begin(), vectorInt.end(),
        [](const int number) { cout << setw(4) << number; });

    file.setArray(vectorInt);
}
```

3. Створити макрос *assert* з параметрами:

– умова;

– повідомлення про помилку.

Якщо умова хибна, друкувати повідомлення та за допомогою *std::abort()* завершити роботу програми.

Створити клас-функтор *StringTester*. Перевантажити оператор круглих дужок для тесту рядка(*char**).

Реалізувати тести:

– рядок не вказує на *NULL*;

– рядок не є порожнім;

– довжина рядка більша за 0.

Створити клас *Ticket* з полями:

– повне ім'я(*char**);

– номер рейсу.

Методи класу:

– конструктор з параметрами(ім'я, прізвище, номер);

– деструктор.

Перевантажити оператор(<<). У конструкторі перевіряти на коректність ім'я та прізвище.

Розв'язання задачі 3:

StringTester.h

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

/*
	Макрос assert з параметрами:
	- умова;
	- повідомлення про помилку.
	Щоб завершити програму достроково
	використовується функція abort().
*/
#define assert(condition, message) \
if(!(condition)){printf("Fatal error: %s\n", message); abort();}

//Клас-функтор
class StringTester
{
public:

	//Перевантаження оператора круглих дужок
	void operator()(const char* str) const;
};
```

StringTester.cpp

```
#include "StringTester.h"

//Перевантаження оператора круглих дужок
void StringTester::operator()(const char* str) const
{
```

```
        //Група тестів для масиву char
        assert(str != NULL, "str == NULL");
        assert(*str != '\0', "str == \'\\0\'");
        assert(strlen(str) > 0, "length <= 0");
}
```

Ticket.h

```
#include "StringTester.h"

class Ticket
{
private:

        //Повне ім'я
        char* fullName;

        //Номер рейсу
        int flightNumber;

public:

        //Конструктор з параметрами
        Ticket(const char* firstName,
                const char* surname, const int number);

        //Деструктор
        ~Ticket();

        //Перевантаження оператора(<<)
        friend ostream& operator<<(ostream& out,
                const Ticket& ticket);
};
```

Ticket.cpp

```
#include "Ticket.h"

//Конструктор з параметрами
Ticket::Ticket(const char* firstName, const char* surname,
                                        const int number)
{
```

```
        //Тести для ім'я та прізвища
        StringTester testString;
        testString(firstName);
        testString(surname);

        //Розмір рядка з повним ім'ям
        int size = strlen(firstName) + strlen(surname) + 2;
        this->fullName = new char[size];

        //Копіювання ім'я
        for (int i = 0; *(firstName + i) != '\0'; i++)
        {
                *(fullName + i) = *(firstName + i);
        }

        fullName[strlen(firstName)] = ' ';

        //Копіювання прізвища
        for (int i = strlen(firstName) + 1;
                                        *(surname) != '\0'; i++)
        {
                *(fullName + i) = *(surname++);
        }

        //На кінець рядка ставимо \0
        fullName[size - 1] = '\0';

        this->flightNumber = number;
}

//Деструктор
Ticket::~Ticket()
{
        delete[] this->fullName;
}

//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const Ticket& ticket)
{
        return out << "Ім'я: " << ticket.fullName
                << ", номер рейсу: " << ticket.flightNumber;
}
```

Practice_task_3.cpp

```
#include <Windows.h>
#include "Ticket.h"

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	Ticket firstTicket("Денис", "Паржницький", 15);
	cout << " --- Перший квиток ---\n";
	cout << firstTicket << endl;

	Ticket secondTicket("", "Пишкін", 21);
	cout << " --- Другий квиток ---\n";
	cout << secondTicket << endl;
}
```

4. Створити клас *Time* з полями:

– години;

– хвилини;

– секунди.

Методи класу:

– конструктор з параметрами за замовчуванням;

– група сетерів класу.

Перевантаження:

– оператор(<<);

– оператор(==) для порівняння часу.

У конструкторі та сетерах передбачити виняткові ситуації, обробити їх за допомогою *assert*.

Розв'язання задачі 4:

Time.h

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

class Time
{
private:
```

```
        //Години
        int hours;

        //Хвилини
        int minutes;

        //Секунди
        int seconds;

public:

        //Конструктор з параметром за замовчуванням
        Time(const int hours = 0,
             const int minutes = 0,
             const int seconds = 0);

        //Група сетерів класу
        Time& setHours(const int hours);
        Time& setMinutes(const int minutes);
        Time& setSeconds(const int seconds);

        //Перевантаження оператора(<<)
        friend ostream& operator<<(ostream& out,
                                   const Time& time);

        //Перевантаження оператора(==) для порівняння часу
        bool operator==(const Time& time) const;
};
```

Time.cpp

```
#include "Time.h"

//Конструктор з параметром за замовчуванням
Time::Time(const int hours, const int minutes,
                              const int seconds)
{

        //За допомогою assert перевіряємо
        //коректність даних
        assert(hours <= 23 && hours >= 0);
        assert(hours <= 59 && hours >= 0);
        assert(seconds <= 59 && seconds >= 0);
```

```
	this->hours = hours;
	this->minutes = minutes;
	this->seconds = seconds;
}

//Група сетерів класу
Time& Time::setHours(const int hours)
{
	assert(hours <= 23 && hours >= 0);

	this->hours = hours;

	//Повертаємо поточний об'єкт
	//Для ланцюжка викликів методів
	return *this;
}


Time& Time::setMinutes(const int minutes)
{
	assert(hours <= 59 && hours >= 0);

	this->minutes = minutes;

	return *this;
}

Time& Time::setSeconds(const int seconds)
{
	assert(seconds <= 59 && seconds >= 0);

	this->seconds = seconds;

	return *this;
}

//Перевантаження оператора(==) для порівняння часу
bool Time::operator==(const Time& time) const
{
	return hours == time.hours &&
		minutes == time.minutes &&
		seconds == time.seconds;
}
```

```
//Перевантаження оператора(<<)
ostream& operator<<(ostream& out, const Time& time)
{
	return out << time.hours   << ":"
		   << time.minutes << ":"
		   << time.seconds;
}
```

Practice_task_4.cpp

```
#include <Windows.h>
#include "Time.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Time firstTime(12, 56, 41);
    Time secondTime(22, 45, 20);

    cout << " --- Друк інформації ---\n";
    cout << "Перший час: " << firstTime << endl;
    cout << "Другий час: " << secondTime << endl;

    if (firstTime == secondTime) cout << "\nЧас збігається\n";
    else cout << "\nЧас не збігається\n";

    firstTime.setHours(25).setMinutes(60).setSeconds(55);
}
```

5. Створити клас *Number* з полем значення(*int*). Методи класу:

– конструктор з параметром;

– конструктор копіювання;

– функції доступу до поля класу;

– функція розрахунку факторіалу числа, передбачити виключення для переповнення.

Перевантаження:

– оператор присвоювання;

– оператор(*), виключення для множення на 0 та 1;

– оператор(+);

– оператор(/), передбачити виключення для ділення на 0.

Усі винятки обробляти за допомогою *assert*.

Розв'язання задачі 5:

Number.h

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

class Number
{
private:

	int value;

public:

	//Конструктор з параметром
	Number(const int value);

	//Конструктор копіювання
	Number(const Number& number);

	//Функції доступу до поля класу
	void setValue(const int value);
	int getValue() const;

	//Перевантаження оператора присвоювання
	Number& operator=(const Number& number);

	//Группа перевантажень арифметичних операторів
	friend Number operator*(const Number& number,
				const int value);
	friend Number operator+(const Number& number,
				const int value);
	friend Number operator/(const Number& number,
				const int value);

	//Функція розрахунку факторіалу числа
	int getFactorial() const;
};
```

Number.cpp

```
#include "Number.h"

//Конструктор з параметром
Number::Number(const int value)
{
	this->value = value;
}

//Конструктор копіювання
Number::Number(const Number& number)
{
	this->value = number.value;
}

//Сетер поля класу
void Number::setValue(const int value)
{
	this->value = value;
}

int Number::getValue() const
{
	return this->value;
}

//Перевантаження оператора присвоювання
Number& Number::operator=(const Number& number)
{
	this->value = number.value;

	return *this;
}

//Функція розрахунку факторіалу числа
int Number::getFactorial() const
{
	int result = 1;
	int n = this->value;

	while (n > 1)
	{
		assert(result <= INT_MAX / n);
		result *= n;
		n--;
	}
```

```
	return result;
}

//Группа перевантажень арифметичних операторів
Number operator*(const Number& number, const int value)
{
	assert(value != 0 && value != 1);

	return Number(number.value * value);
}

Number operator+(const Number& number, const int value)
{
	return Number(number.value + value);
}

Number operator/(const Number& number, const int value)
{
	assert(value != 0);

	return Number(number.value / value);
}
```

Practice_task_5.cpp

```
#include <Windows.h>
#include "Number.h"
using namespace std;

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Number number(5);

    //Арифметичні перетворення
    number = number + 5;
    number = number * 2;
    number = number / 4;

    cout << "Значення: " << number.getValue();
    cout << "\nФакторіал числа: " << number.getFactorial()
                                << endl;
```

```
    //Змінюємо значення на 15
    number.setValue(15);

    //Помилка, після числа 12 відбувається переповнення int
    cout << "\nФакторіал числа: " << number.getFactorial();
}
```

Контрольні запитання

1. Що таке *assert*? Навіщо він використовується?
2. Коли потрібно використовувати макрос *assert*?
3. Чи працює макрос *assert* у *release* конфігурації програми?
4. Які є способи використання макросу *assert* у функціях?
5. Що відбувається з програмою після успішного виконання макросу *assert*?

14. КЛАС STRING

Мета: Оволодіння практичними навичками роботи з рядковим класом в мові С++

***Теми для попереднього опрацьовування*:**

- Рядки в мові С.
- Функції в мовах С/С++.
- Перевантаження операцій.
- Контейнерні класи.
- Алгоритми STL.

Теоретичні відомості

Стандартна бібліотека С++ містить багато корисних класів, одним з яких є *string*. Це рядковий клас, який дозволяє виконувати операції присвоювання, порівняння і модифікації рядків.

С++ рядки визначені в бібліотеці <*string*>, яку потрібно підключити за допомогою директиви

```
# include <string>
```

Функціонал string

Створення і видалення

Рядкові класи мають ряд конструкторів і деструктор, які можна використовувати для створення рядків.

```
string s;                    // порожній рядок
string s(cstr);              // створює рядок із С-рядка
string s(cstr, len);         // створює рядок з len
                             // символів С-рядка
string s(num, ch);           // створює рядок з num
                             // символів ch
string s(str);               // створює рядок з рядка str
```

Розмір і ємність

capacity() – повертає кількість символів, які рядок може зберігати без додаткового перевиділення пам’яті

```
string sString("0123456789");
cout << "Length: " << sString.length();
cout << "Capacity: " << sString.capacity();
                                          // Length: 10
                                          // Capacity: 15
```

empty() – повертає логічне значення, яке вказує, чи є рядок порожнім

```
string sString1("Not Empty");
cout << (sString1.empty() ? "true" : "false");   // false
string sString2;                                  // пустий рядок
cout << (sString2.empty() ? "true" : "false");   // true
```

length(), *size*() – повертають кількість символів в рядку

```
string sSomething("012345");
cout << sSomething.length();                          // 6
```

max_size() – повертає максимальний розмір рядка, який може бути виділений

```
string sString("MyString");
cout << sString.max_size();                    // 2147483647
```

reserve() – розширює або зменшує ємність рядка.

```
string sString("0123456789");
cout << "Length: " << sString.length();
cout << "Capacity: " << sString.capacity();

sString.reserve(300);
cout << "Length: " << sString.length();
cout << "Capacity: " << sString.capacity();
                                          // Length: 10
                                          // Capacity: 15
                                          // Length: 10
                                          // Capacity: 303
```

Доступ до елементів

[ ], *at*() – доступ до елемента по заданому індексу

```
string sSomething("abcdefg");
cout << sSomething[4];                          // e
sSomething[4] = 'A';
sSomething.at(4) = 'A';
cout << sSomething;                             // abcdAfg
```

Модифікація рядків

=, *assign*() – присвоюють нове значення рядку

```
// Присвоюємо std::string інший рядок
sString = string("One");
cout << sString << endl;                      // One

const string sTwo("Two");
sString.assign(sTwo);
cout << sString << endl;                      // Two
```

+=, *append*(), *push_back*() – додають символи до кінця рядка

```
string sString("one");
sString += string(" two");
string sThree(" three");
sString.append(sThree);
cout << sString;                           // one two three

string sString("one ");
const string sTemp("twothreefour");
sString.append(sTemp, 8, 4);
         // додаємо до std::string підрядок sTemp довжиною
         // 4, починаючи з символа під індексом 8
cout << sString;                           // one four

string sString("two");
sString += ' ';
sString.push_back('3');
cout << sString;                        // two 3
```

insert() – вставляє символи в довільний індекс рядка

```
string sString("bbb");
cout << sString;                                  // bbb
sString.insert(2, std::string("mmm"));
cout << sString;                                  // bbmmmb
sString.insert(5, "aaa");
cout << sString;                                  // bbmmmaaab
```

clear() – видаляє всі символи рядка

```
s1.clear();
```

erase() – видаляє символи за довільним індексом рядка

```
string s1("NEW STRING");
s1.erase(0,4);                                    //s1=STRING
```

replace() – замінює символи довільних індексів рядка іншими символами

```
string s1("NEW STRING"), s2("******");
s1.insert(3,s2);
cout << "s1=" << s1 << endl;       //s1=NEW****** STRING
string s3("OLD$$$");
s1.replace(0,3,s3);
cout << "s1=" << s1 << endl;      //s1=OLD$$$****** STRING
s2.replace(1,3,s3,4,2);
cout << "s2=" << s2 << endl;      //s2=*$$**
```

resize() – розширює або зменшує рядок (видаляє або додає символи в кінці рядка)

```
std::string myString = "Hello, world!";
// Зміна розміру рядка на 5 символів
myString.resize(5);
cout << myString;                        // Hello
// Зміна розміру рядка на 10 символів, нові символи будуть
// нульовими символами
myString.resize(10);
std::cout << myString;                   // Hello□□□□□
// Зміна розміру рядка на 8 символів і заповнення
// нових символів ''x''
```

```
myString.resize(8, 'X');
cout << myString;                    // HelloXXX
```

swap() – міняє місцями значення двох рядків

```
string sStr1("green");
string sStr2("white");
cout << sStr1 << " " << sStr2 << endl;    // green white
swap(sStr1, sStr2);
cout << sStr1 << " " << sStr2 << endl;    // white green
sStr1.swap(sStr2);
cout << sStr1 << " " << sStr2 << endl;    // green white
```

Ввід/вивід

>>, *getline*() – зчитують значення з вхідного потоку в рядок

<< – записує значення рядка у вихідний потік

Конвертація std::string в рядки C-style

c_str() – конвертує рядок в рядок C-style з нуль-термінатором в кінці

```
string sSomething("abcdefg");
cout << strlen(sSomething.c_str());                  // 7
```

copy() – копіює вміст рядка (без нуль-термінатора) в масив типу *char*

```
string sSomething("lorem ipsum dolor sit amet");
char szBuf[20];
int nLength = sSomething.copy(szBuf, 5, 6);
szBuf[nLength] = '\0';        // завершуємо рядок в буфері
cout << szBuf;                // ipsum
```

data() – повертає вміст рядка у вигляді масиву типу *char*, який не закінчується нуль-термінатором

```
string sSomething("abcdefg");
const char *szString = "abcdefg";
// Функція memcmp() порівнює два вищенаведених рядки
// C-style і повертає 0, якщо вони рівні
if (memcmp(sSomething.data(), szString,
           sSomething.length()) == 0)
      cout << "The strings are equal";
```

```
else
      cout << "The strings are not equal";
                                  // The strings are equal
```

Порівняння рядків

==, != – порівнюють, чи є два рядки рівними/нерівними (повертають значення типу *bool*)

```
string  s5("new string");
cout << "s5=" << s5 << endl;          // s5=new string
string  s6(s5);
cout << "s6=" << s6 << endl;          // s6=new string
cout << s5 == s6 << endl;             // 1
```

<, <=, >, >= – порівнюють, чи є два рядки менше або більше один одного (повертають значення типу *bool*) string s3(str1, 3);

```
string s3("old string", 3);
cout << "s3=" << s3 << endl;          // s3=old
string  s4(5, '5');
cout << "s4=" << s4 << endl;          // s4=55555
cout << s3 > s4 << endl;             // 1
cout << s3 < s4 << endl;             // 0
```

compare() – порівнює, чи є два рядки рівними/нерівними (повертає -1, 0 або 1)

```
string str1 = "apple";
string str2 = "banana";
int result = str1.compare(str2);
if (result < 0) {
  cout << "str1 is less than str2" << endl;
    } else
       if (result > 0) {
        cout << "str1 is greater than str2" << endl;
       } else
        cout << "str1 is equal to str2" << endl;
}
```

Підрядки і конкатенація

+ – з'єднує два рядки

```
string str1 = "Hello, ";
string str2 = "world!";
string combined = str1 + str2;
cout << combined << endl;             // Hello, world!
```

substr() – повертає підрядок.

```
string original = "Hello, world!";
string sub1 = original.substr(7);
string sub2 = original.substr(0, 5);
cout << "sub1: " << sub1 << endl;              // world!
cout << "sub2: " << sub2 << endl;              // Hello
```

Пошук

find() – шукає індекс першого символу/підрядка

```
string sentence = "The quick brown fox";
size_t found = sentence.find("fox");
if (found != string::npos) {
  cout << "Substring found at position: " << found;
    } else {
  cout << "Substring not found." << endl;
    }
```

find_first_of () – шукає індекс першого символу з набору символів

find_first_not_of() – шукає індекс першого символу НЕ з набору символів

find_last_of () – шукає індекс останнього символу з набору символів

find_last_not_of() – шукає індекс останнього символу НЕ з набору символів

rfind() – шукає індекс останнього символу/підрядка.

***Приклад* 1.** Робота з рядками. Знайти кількість входжень підрядка *S*1 в рядок *S*.

```
string s, s1;
int k, i;
cin >> s;                             // 123123123
```

```
cin >> s1;                              // 123
k = 0;
i = 0;
while (s.find(s1, k) != -1)
{
    k = s.find(s1, k) + s1.length();
    i++;
}
cout << i;                              // 3
```

***Приклад* 2.** Робота з рядками. Із заданого символьного рядка вибрати ті символи, які зустрічаються в ній лише один раз, в тому порядку, в якому вони зустрічаються в тексті.

```
string s;
bool f;
getline(cin, s);                        // 112342053
for (int i = 0; i < s.size(); i++)
{
   // вважаємо, що черговий символ зустрічається один раз
   f = true;
   for (int j = 0; j < s.size() && f; j++)
      f = !(s[i] == s[j] && i != j);
   if (f)
     cout << s[i] << " ";               // 4 0 5
}
```

***Приклад* 3.** Робота з рядками. Використати клас *string* для створення асоціативного контейнера. Використати лямбда-функції. Здійснити пошук по ключу.

```
map <string, string> m;
m.insert({ "yes","no" });
m["good"] = "bad";
m["top"] = "bottom";

for_each(m.begin(), m.end(),
     [](const auto& item)
       {cout <<item.first << " " << item.second<<endl; }
  );
```

```
                                        // good bad
                                        // top bottom
                                        // yes yes
    string value;
    cin >> value;                       // top
    auto p = find_if(m.begin(), m.end(),
         [&value](const auto& item)
            {return item.first == value; }
      );

    if (p != m.end())
       cout << p->second << endl;       // bottom
    else
       cout << "not found";
```

Задачі

Загальні умови

Кожне завдання містить приклад використання рядків класу *String* та функцій їх обробки в мові С++, і може містити умови вирішення. Після формулювання завдання надано рішення у вигляді програмного коду.

1. Створити клас *WordCounter*, який рахує кількість слів у файлі, заданого користувачем. Передбачити ситуацію, в якій шукане слово закінчується розділовим знаком:

– кома;

– крапка;

– знак питання;

– знак оклику.

Розв'язання задачі 1:

WordCounter.h

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

class WordCounter
```

```
{
private:

	//Функція порівняння поточного слова з шуканим
	bool equal(const string& curWord,
		   const string& word) const;
public:

	//Функція підрахунку кількості слів
	int getCount(const string& path,
		     const string& word) const;
};
```

WordCounter.cpp

```
#include "WordCounter.h"

//Функція порівняння поточного слова з шуканим
bool WordCounter::equal(const string& curWord,
                        const string& word) const
{

    //Розділові знаки
    char symbols[] = { ',', '.', '?', '!' };

    //Якщо слова рівні, повертаємо true
    if (curWord == word) return true;

    for (int i = 0; i < sizeof(symbols); i++)
        //Додаємо до шуканого слова розділові знаки
        //Якщо є збіг повертаємо true
        if (curWord == word + symbols[i]) return true;

    //Якщо слово не знайшли повертаємо false
    return false;
}

int WordCounter::getCount(const string& path,
                          const string& word) const
{
    ifstream fin(path);
    if (!fin.is_open())
        throw exception("Помилка відкриття файлу!");

    string curWord; //Поточне слово з файлу
```

```
    int count = 0; //Кількість слів

    while (!fin.eof())
    {
        fin >> curWord;
        if (this->equal(curWord, word)) count++;
    }

    fin.close();
    return count;
}
```

Practice_task_1.cpp

```
#include <Windows.h>
#include "WordCounter.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    WordCounter wordCounter;
    string word;
    const string path = "text.txt";

    try
    {
        cout << "Введіть слово для пошуку: ";
        cin >> word;

        cout << "Кількість шуканих слів у файлі: "
             << wordCounter.getCount(path, word);
    }
    catch (const exception& exception)
    {
        cout << exception.what() << endl;
    }
}
```

2. Створити клас-функтор *StringFilter*, який шукає символи, які зустрічаються лише раз у тексті. Знайдені символи записати у файл в зворотному порядку.

Розв'язання задачі 2:

StringFilter.h

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

//Клас-функтор
class StringFilter
{
public:

	//Перевантаження оператора круглих дужок
	void operator()(const string& path,
			const string& text) const;
};
```

StringFilter.cpp

```
#include "StringFilter.h"

void StringFilter::operator()(const string& path,
				const string& text) const
{
	//Відкриття файлу на запис
	ofstream fout(path);
	if (!fout.is_open())
		throw exception("Помилка відкриття файлу");

	//Прапорець символу, що зустрічається
	//0 - зустрівся більше одного разу
	//1 - зустрівся один раз
	bool flag;

	//Допоміжний рядок
	string tempStr;

	for (int i = 0; i < text.size(); i++)
	{
		//Гіпотеза, що символ зустрівся лише раз
		flag = 1;

		for (int j = 0; (j < text.size()) && flag; j++)
```

```
                {
                        //Якщо символ збігся повертаємо
                        //0, виходимо з циклу
                        flag = !(text[i] == text[j] && i != j);
                }

                //Якщо прапорець = 1, дозаписуємо символ
                //у допоміжний рядок
                if (flag)
                        tempStr += text[i];
        }

        //Записуємо рядок у зворотному порядку
        for (int i = tempStr.size() - 1; i >= 0; i--)
        {
                fout << tempStr[i];
        }

        fout.close();
}
```

Practice_task_2.cpp

```
#include <Windows.h>
#include "StringFilter.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    StringFilter filter;
    string text = "fffgdff213123hp";
    const string path = "resultText.txt";

       try
       {
              filter(path, text);
       }
       catch (const exception& exception)
       {
              cout << exception.what() << endl;
       }
}
```

3. Створити клас *Notes*, який частково імітує роботу однойменного додатку. Поля класу:

– пароль;

– прапорець входу до облікового запису(*bool*);

– нотатки(*map<int, string>).*

Методи класу:

– конструктор з параметром(пароль);

– функція входу до облікового запису, якщо паролі збігаються, змінюємо прапорець;

– функція виходу з облікового запису;

– функція зміни паролю(кількість символів більше 7);

– функція додавання нотатку, де ключ номер нотатки, а значення текст нотатки;

– функція заміни тексту в нотатці, користувач вводить номер нотатки, текст, індекс початку заміни, кількість символів для заміни(*replace*);

– функція друку нотаток.

Передбачити ситуацію, якщо користувач не ввійшов в акаунт, він не може користуватися нотатками.

Розв'язання задачі 3:

Notes.h

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

class Notes
{
private:

	//Пароль
	string password;

	//Прапорець входу до облікового запису
	//1 - користувач увійшов у систему
	//0 - користувач не увійшов
```

```
        bool enterFlag;

        //нотатки
        map<int, string> notes;

public:

        //Конструктор з параметрами
        Notes(const string& password);

        //Функція входу до облікового запису
        void login(const string& password);

        //Функція виходу з облікового запису
        void exit();

        //Функція зміни паролю
        void setPassword(const string& newPassword);

        //Функція додавання нотатки
        void addNote(const string& note);

        //Функція заміни тексту в нотатці, аргументи:
        // - номер нотатки
        // - новий текст
        // - початкова позиція для заміни
        // - кількість змінюваних символів
        void changeNote(const int index, const string& text,
                        const int deletePos, const int count);

        //Друк нотаток
        void printNotes() const;
};
```

Notes.cpp

```
#include "Notes.h"

//Конструктор з параметрами
Notes::Notes(const string& password)
{
        this->password = password;
        this->enterFlag = 0;
}
```

```
//Функція входу  до облікового запису
void Notes::login(const string& password)
{
	//Якщо паролі збігаються, змінюємо прапорець
	if (this->password == password)
		this->enterFlag = 1;
	else
		throw exception("Невірний пароль");
}

//Функція виходу з облікового запису
void Notes::exit()
{
	this->enterFlag = 0;
}

//Функція зміни паролю
void Notes::setPassword(const string& newPassword)
{
	if (!enterFlag)
		throw exception("Ви не увійшли до системи");

	if (newPassword.size() < 8) {
		throw exception("Пароль повинен містити більше 7
                                                символів!");
	}
	else {
		this->password = newPassword;
	}
}

//Функція додавання нотатки
void Notes::addNote(const string& note)
{
	if (!enterFlag)
		throw exception("Ви не увійшли до системи");

	notes.insert({ notes.size() + 1, note });
}

//Функція зміни нотатки
void Notes::changeNote(const int pos, const string& text,
                       const int deletePos, const int count)
{
	if (!enterFlag)
```

```
            throw exception("Ви не увійшли до системи");

        if (pos < 1 && pos > notes.size())
            throw exception("Некоректний індекс");

        // Видаляємо потрібну кількість символів, на
        // їх місце вставляємо текст
        notes.find(pos)->second.replace(deletePos, count,
                                        text, 0, text.size());
}

//Друк нотаток
void Notes::printNotes() const
{
        if (!enterFlag)
            throw exception("Ви не увійшли до системи");

        cout << " --- Мої нотатки ---\n";
        for (auto iter = notes.begin(); iter != notes.end();
                                                    iter++)
        {
            cout << iter->first << ". "
                 << iter->second << endl;
        }
}
```

Practice_task_3.cpp

```
#include <Windows.h>
#include "Notes.h"

int main()
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);

        string myPassword = "12adfG521";
        Notes account(myPassword);

        try
        {
            cout << "Введіть пароль: ";
            cin >> myPassword;
            account.login(myPassword);
```

```
            //Додавання нотаток
            account.addNote("Тренування 16:00, взяти форму");
            account.addNote("Купити продукти");
            account.addNote("Замовити подарунок");

            //Зміна часу у першій замітці
            account.changeNote(1, "17:00", 11, 5);

            cout << "Зміна паролю(кількість символів > 7): ";
            cin >> myPassword;
            account.setPassword(myPassword);

            //Друк нотаток
            account.printNotes();

            //Вихід з облікового запису
            account.exit();
        }
        catch (const exception& exception)
        {
            cout << exception.what() << endl;
        }
}
```

4. Створити клас-функтор *TextTransformer*, який у кожному слові файлу шукає задану літеру і виділяє її лапками. Параметрами перевантаженого оператора круглих дужок є:

– шлях до файлу з початковим текстом;

– шлях до файлу зі зміненим текстом.

У класі наявне поле символ пошуку. Передбачити сетер для поля класу.

Розв'язання задачі 4:

TextTransformer.h

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

class TextTransformer
```

```
{
private:

	//Символ пошуку
	char searchSymbol;

	//Функція для зміни слова
	void transformWord(string& word) const;

public:

	//Конструктор з параметром
	TextTransformer(const char searchSymbol);

	//Сетер поля класу
	void setSymbol(const char searchSymbol);

	//Перевантаження оператора круглих дужок
	void operator()(const string& inputFilePath,
			const string& outputFilePath) const;
};
```

TextTransformer.cpp

```
#include "TextTransformer.h"

//Конструктор з параметром
TextTransformer::TextTransformer(const char searchSymbol)
{
	this->searchSymbol = searchSymbol;
}

//Сетер поля класу
void TextTransformer::setSymbol(const char searchSymbol)
{
    this->searchSymbol = searchSymbol;
}

//Функція для зміни слова
void TextTransformer::transformWord(string& word) const
{
    int index = 0;

    //Шукаємо індекс символу
    while ((index = word.find(searchSymbol, index)) != -1)
```

```
    {
        //Перед та після символу вставляємо лапки
        word.insert(index, "\'");
        index += 2;
        word.insert(index, "\'");
    }
}

//Перевантаження оператора круглих дужок
void TextTransformer::operator()(const string& inputFilePath,
                                 const string& outputFilePath) const
{
    ifstream fin(inputFilePath);
    ofstream fout(outputFilePath);

    if (!fin.is_open() || !fout.is_open())
        throw exception("Помилка відкриття файлу");

    string word;
    while (!fin.eof())
    {
        fin >> word;

        this->transformWord(word);

        fout << word << " ";
    }

    fin.close();
    fout.close();
}
```

Practice_task_4.cpp

```
#include <Windows.h>
#include "TextTransformer.h"

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    TextTransformer transformer('a');

       //Шлях до файлу з початковим текстом
```

```
        string inputPath = "inputText.txt";

        //Шлях до файлу зі зміненим текстом
        string outputPath = "outputText.txt";

        try
        {
                transformer(inputPath, outputPath);
        }
        catch (const exception& exception)
        {
                cout << exception.what() << endl;
        }
}
```

Контрольні запитання

1. Якими способами можна ініціалізувати об'єкт класу *string*?
2. Які операції можна робити з рядками?
3. Які переваги у класу *string* у порівнянні зі стандартним рядком? Які недоліки?
4. Перелічіть основні методи класу *string*. Дайте коротке пояснення кожному.
5. Чи є різниця між методом *size* та *length* у класі *string*?
6. Чи ставиться нуль-термінатор після застосування методу *copy*?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Herbert Schildt. C++: A Beginner's Guide, Second Edition. – McGraw Hill; 2nd edition (December 3, 2008). ISBN-13 : 978-0072232158. – 2003.– 576 p.
2. Bjarne Stroustrup. The C++ Programming Language. – Addison-Wesley Professional; 4th edition (May 9, 2013). ISBN-13 : 978-0275967307 – 1376 p.
3. Robert Lafore. Object Oriented Programming in C++. – Sams; Subsequent edition (December 29, 2015). ISBN-13 : 978-0672323089. – 1012 p.
4. Джордж Хайнеман, Гері Полліс, Стенлі Селков. Алгоритми. Довідник з прикладами на C, C++, Java і Python.- Діалектика, 2017. 432 с.
5. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns. Elements of Reusable Object-Oriented Software. – Pearson Education India; 1st edition (January 1, 2015). – 417 p.
6. Об'єктно-орієнтоване програмування мовою C++ : навчальний посібник. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 404 с.
7. https://www.ibm.com/developerworks/ru/library/l-git-subversion-1/index.html - Git для пользователей Subversion.
8. https://try.github.io/ - Resources to learn Git.
9. https://guides.github.com/activities/hello-world/ - The Hello World project.
10. https://githowto.com/ru - Git How To.
11. http://gitimmersion.com/ - Git Immersion. A guided tour.
12. https://www.eclipse.org/documentation/ - Eclipse Documentation.
13. https://www.jetbrains.com/idea/documentation/ - IntelliJ IDEA. Learn and support.
14. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow - Gitflow Workflow.
15. https://nvie.com/posts/a-successful-git-branching-model/ - A successful Git branching model

ЗМІСТ

Навчальне видання

ЛЮБЧЕНКО Наталія Юріївна
СОБОЛЬ Максим Олегович
ПОДОРОЖНЯК Андрій Олексійович
ПУГАЧОВ Роман Володимирович
ЛЮБЧЕНКО Олексій Вікторович

ОСНОВИ ООП С++ В ПРИКЛАДАХ
Частина 2

Навчально-методичний посібник
для студентів комп'ютерних спеціальностей
вищих навчальних закладів

Відповідальний за випуск проф. Александрова Т. Є.

Роботу рекомендувала до видання доц. Костюк О. В.

В авторській редакції

План 2026 р., поз. 20

Гарнітура Times New Roman. Ум. друк. арк. 5,06

Видавничий центр НТУ «ХПІ»
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2.

Електронне видання