**Міністерство освіти і науки України**

**Національний університет “Львівська політехніка”**

**Інститут комп’ютерних наук та інформаційних технологій**

***Кафедра ПЗ***

**Звіт**

до лабораторної роботи №9

на тему « Шаблони функцій та класів »

з дисципліни “Об’єктно-орієнтоване програмування”

**Виконав:**

ст. гр. ПЗ-14

Попович Д.О.

**Прийняла:**

ст. викл. каф. ПЗ

Дяконюк Л.М.

12.05.2023 р.

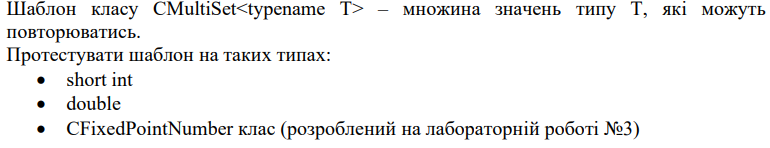
Львів – 2023

**Тема роботи:** Шаблони функцій та класів.

**Мета роботи:** Навчитись створювати шаблони функцій та класів, використовувати різні види параметрів шаблонів, створювати екземпляри шаблонів.

**Завдання**

1. Змінити реалізацію класу із лабораторної роботи №8 відповідно до варіанту, зробивши його шаблоном класу.
2. Переконатись, що у класі є визначений оператор порівняння (==).
3. Переконатись у правильності роботи різних інстанціацій шаблону відповідно до варіанту:



1. Продемострувати роботу шаблону класу таким чином (на вибір):
2. За допомогою тестів Google Test.
3. За допомогою інтерактивної програми із демострацією її роботи.
4. Написати шаблон функції FindElementInArray(), яка буде приймати масив елементів будьякого типу та елемент, який потрібно знайти в цьому масиві. Функція повинна повернути індекс першого входження елемента в масив, або розмір масиву, якщо елемент не знайдено. Протестувати функцію на будь-якому вбудованому типі, а також розробленому відповідно до п.1 класі.

6. Оформити звіт до лабораторної роботи.

**Теоретичні відомості**

Шаблони є основою для узагальненого програмування на C++. Як строго типізована мова, C++ вимагає, щоб усі змінні мали певний тип, або явно оголошений програмістом, або виведений компілятором. Однак багато структур даних і алгоритмів виглядають однаково незалежно від того, з яким типом вони працюють. Шаблони дозволяють визначати операції класу чи функції, а користувачу дозволяють вказувати, з якими конкретними типами ці операції мають працювати. По суті шаблон визначає:

• родину класів (шаблон класу)

• родину функцій (шаблон функцій)

• псевдонім до родини типів (шаблон-псевдонім) – починаючи із C++11

• родину змінних (шаблон змінних) – починаючи із C++14

Шаблони мають один або кілька параметрів. Існує три види параметрів:

1. параметри шаблону, що позначають тип (type template parameters);
2. параметри шаблону, що позначають не тип (non-type template parameters);
3. шаблонні параметри шаблонів (template template parameters).

Шаблон — це конструкція, яка ґенерує звичайний тип, функцію або змінну (починаючи із С++11) під час компіляції на основі аргументів, які користувач надає для параметрів шаблону. Параметр шаблону може мати будь-яку назву, хоча часто використовують одну велику літеру. Ключове слово typename означає, що цей параметр позначає тип. Замість typename також може бути ключове клово class. В даному контексті вони мають однакове значення. Коли функція викликається, компілятор замінить кожен екземпляр T назвою конкретного типу, який або вказано користувачем, або виведений компілятором. Процес генерації класу, функції чи змінної з шаблону називається інстанціюванням шаблону.

**Вміст файлів**

Вміст файлу CMultiSet.h:

#pragma once

#include <iostream>

#include <stdexcept>

using namespace std;

template <typename T>

class CMultiSet {

private:

int size;

T\* values;

public:

CMultiSet();

CMultiSet(int capacity);

CMultiSet(const CMultiSet& other);

CMultiSet(CMultiSet&& other) noexcept;

~CMultiSet();

T getElement(int index) const noexcept;

int countOfCertainElement(const T& value) const noexcept;

int cardinality() const noexcept;

T findMax() const;

T findMin() const;

T findSum() const;

T printInEndOrder() const noexcept;

void addElement(const T& temp);

void deleteElement(const T& temp);

void readArray();

void printArray();

CMultiSet<T> operator+(const CMultiSet<T>& other) const noexcept;

CMultiSet<T> operator-(const CMultiSet<T>& other) const noexcept;

CMultiSet<T> operator/(const CMultiSet<T>& other) const noexcept;

CMultiSet<T>& operator=(const CMultiSet<T>& other);

CMultiSet<T>& operator=(CMultiSet<T>&& other) noexcept;

template <typename T>

friend istream& operator>>(istream& input, CMultiSet<T>& other) noexcept;

template <typename T>

friend ostream& operator<<(ostream& output, CMultiSet<T>& other) noexcept;

};

template<typename T>

CMultiSet<T>::CMultiSet()

: values(nullptr), size(0) { }

template<typename T>

CMultiSet<T>::CMultiSet(int capacity)

: values(nullptr) {

this->size = capacity;

values = new T[size];

if (!values) {

throw bad\_alloc();

}

for (int i = 0; i < size; i++)

{

values[i] = T();

}

}

template<typename T>

CMultiSet<T>::CMultiSet(const CMultiSet& other) {

size = other.size;

values = new T[size];

if (!values) {

throw bad\_alloc();

}

for (int i = 0; i < size; i++) {

values[i] = other.values[i];

}

}

template<typename T>

CMultiSet<T>::CMultiSet(CMultiSet&& other) noexcept {

size = other.size;

values = other.values;

other.size = 0;

other.values = nullptr;

}

template<typename T>

CMultiSet<T>::~CMultiSet() {

delete[] values;

}

template<typename T>

T CMultiSet<T>::getElement(int index) const noexcept {

return values[index];

}

template<typename T>

int CMultiSet<T>::countOfCertainElement(const T& value) const noexcept {

int count = 0;

for (int i = 0; i < size; i++) {

if (values[i] == value)

count++;

}

return count;

}

template<typename T>

int CMultiSet<T>::cardinality() const noexcept {

return size;

}

template<typename T>

T CMultiSet<T>::findMax() const {

if (size == 0) {

throw runtime\_error("Empty multiset");

}

else {

T max = values[0];

for (int i = 0; i < size; i++) {

if (values[i] > max) {

max = values[i];

}

}

return max;

}

}

template<typename T>

T CMultiSet<T>::findMin() const {

if (size == 0) {

throw runtime\_error("Empty multiset");

}

else {

T min = values[0];

for (int i = 0; i < size; i++) {

if (values[i] < min) {

min = values[i];

}

}

return min;

}

}

template<typename T>

T CMultiSet<T>::findSum() const {

if (size == 0) {

throw runtime\_error("Empty multiset");

}

else {

T sum = 0;

for (int i = 0; i < size; i++) {

sum = sum + values[i];

}

return sum;

}

}

template<typename T>

void CMultiSet<T>::addElement(const T& temp) {

size++;

T\* newValues = new T[size];

if (!newValues) {

throw bad\_alloc();

}

for (int i = 0; i < size - 1; i++) {

newValues[i] = values[i];

}

delete[] values;

values = newValues;

values[size - 1] = temp;

for (int i = 0; i < size; i++) {

for (int j = 0; j < size - i - 1; j++) {

if (values[j] > values[j + 1]) {

T temp = values[j];

values[j] = values[j + 1];

values[j + 1] = temp;

}

}

}

}

template<typename T>

void CMultiSet<T>::deleteElement(const T& temp) {

int index = -1;

for (int i = 0; i < size; i++) {

if (values[i] == temp) {

index = i;

break;

}

}

if (index == -1) {

throw runtime\_error("Element not found");

}

size--;

for (int i = index; i < size; i++) {

values[i] = values[i + 1];

}

T\* newValues = new T[size];

if (!newValues) {

throw bad\_alloc();

}

for (int i = 0; i < size; i++) {

newValues[i] = values[i];

}

delete[] values;

values = newValues;

}

template<typename T>

void CMultiSet<T>::readArray() {

int count;

cout << "Enter count: ";

if (!(cin >> count)) {

throw runtime\_error("Bad input");

}

for (int i = 0; i < count; i++) {

T value;

cout << "Enter value " << i + 1 << ": ";

cin >> value;

if (!value) {

throw runtime\_error("Bad input");

}

addElement(value);

}

}

template<typename T>

CMultiSet<T> CMultiSet<T>::operator+(const CMultiSet<T>& other) const noexcept {

CMultiSet<T> multiSet(0);

for (int i = 0; i < size; i++) {

multiSet.addElement(values[i]);

}

for (int i = 0; i < other.cardinality(); i++) {

multiSet.addElement(other.values[i]);

}

return multiSet;

}

template<typename T>

CMultiSet<T> CMultiSet<T>::operator-(const CMultiSet<T>& other) const noexcept {

CMultiSet<T> multiSet(0);

for (int i = 0; i < size; i++) {

T currentElement = values[i];

bool isContains = false;

for (int j = 0; j < other.size; j++) {

if (currentElement == other.values[i]) {

isContains = true;

break;

}

}

if (isContains) {

multiSet.addElement(currentElement);

}

}

return multiSet;

}

template<typename T>

CMultiSet<T> CMultiSet<T>::operator/(const CMultiSet<T>& other) const noexcept {

CMultiSet<T> multiSet(0);

for (int i = 0; i < size; i++) {

T currentElement = values[i];

bool isContains = false;

for (int j = 0; j < other.size; j++) {

if (currentElement == other.values[i]) {

isContains = true;

break;

}

}

if (!isContains) {

multiSet.addElement(currentElement);

}

}

return multiSet;

}

template<typename T>

CMultiSet<T>& CMultiSet<T>::operator=(const CMultiSet<T>& other) {

size = other.size;

values = new T[size];

if (!values) {

throw bad\_alloc();

}

for (int i = 0; i < size; i++) {

values[i] = other.values[i];

}

return \*this;

}

template<typename T>

CMultiSet<T>& CMultiSet<T>::operator=(CMultiSet<T>&& other) noexcept {

size = other.size;

values = other.values;

other.size = 0;

other.values = nullptr;

return \*this;

}

template<typename T>

void CMultiSet<T>::printArray() {

for (int i = size-1; i >= 0 ; i--) {

cout << values[i] << " ";

}

cout << endl;

}

template<typename T>

istream& operator>>(istream& input, CMultiSet<T>& other) noexcept {

other.readArray();

return input;

}

template<typename T>

ostream& operator<<(ostream& output, CMultiSet<T>& other) noexcept {

if (other.size == 0) {

throw runtime\_error("Empty multiset");

}

other.printArray();

return output;

}

Вміст файлу CFixedPointNumber.h:

#pragma once

#include <iostream>

#include <cmath>

class CFixedPointNumber {

private:

int intPart;;

int fractPart;

public:

CFixedPointNumber(int intPart = 0, int fractPart = 0);

void setIntPart(int intPart);

void setFractPart(int fractPart);

void setDoubleValue(double value);

int getIntPart() const;

int getFractPart() const;

double getDoubleValue() const;

CFixedPointNumber findSum(const CFixedPointNumber& value) const;

CFixedPointNumber findSum(int value) const;

CFixedPointNumber findSubstraction(const CFixedPointNumber& value) const;

CFixedPointNumber findSubstraction(int value) const;

CFixedPointNumber findMultiplication(const CFixedPointNumber& value) const;

CFixedPointNumber findMultiplication(int value) const;

CFixedPointNumber findDivision(const CFixedPointNumber& value) const;

CFixedPointNumber findDivision(int value) const;

CFixedPointNumber operator+(const CFixedPointNumber& value) const;

CFixedPointNumber operator-(const CFixedPointNumber& value) const;

CFixedPointNumber operator\*(const CFixedPointNumber& value) const;

CFixedPointNumber operator/(const CFixedPointNumber& value) const;

operator double() const;

friend std::istream& operator>> (std::istream& input, CFixedPointNumber value);

friend std::ostream& operator<< (std::ostream& output, CFixedPointNumber value);

friend bool operator> (const CFixedPointNumber& value, const CFixedPointNumber& otherValue);

friend bool operator< (const CFixedPointNumber& value, const CFixedPointNumber& otherValue);

friend bool operator== (const CFixedPointNumber& value, const CFixedPointNumber& otherValue);

};

Вміст файлу CFixedPointNumber.cpp:

#include "CFixedPointNumber.h"

CFixedPointNumber::CFixedPointNumber(int intPart, int fractPart) {

setIntPart(intPart);

setFractPart(fractPart);

}

void CFixedPointNumber::setIntPart(int intPart) {

this->intPart = intPart;

}

void CFixedPointNumber::setFractPart(int fractPart) {

this->fractPart = fractPart;

}

void CFixedPointNumber::setDoubleValue(double value) {

this->intPart = (int)value;

this->fractPart = (int)((value - intPart) \* 100);

}

int CFixedPointNumber::getIntPart() const {

return intPart;

}

int CFixedPointNumber::getFractPart() const {

return fractPart;

}

double CFixedPointNumber::getDoubleValue() const {

return ((double)intPart + ((double)fractPart / 100));

}

CFixedPointNumber CFixedPointNumber::findSum(const CFixedPointNumber& value) const {

int sumIntPart = this->intPart + value.intPart;

int sumFractPart = this->fractPart + value.fractPart;

if (sumFractPart >= 100) {

sumIntPart++;

sumFractPart -= 100;

}

return CFixedPointNumber(sumIntPart, sumFractPart);

}

CFixedPointNumber CFixedPointNumber::findSum(int value) const {

int sumIntPart = this->intPart + value;

int sumFractPart = this->fractPart;

return CFixedPointNumber(sumIntPart, sumFractPart);

}

CFixedPointNumber CFixedPointNumber::findSubstraction(const CFixedPointNumber& value) const {

int substractIntPart = this->intPart - value.intPart;

int substractFractPart = this->fractPart - value.fractPart;

if (substractFractPart < 100) {

substractIntPart--;

substractFractPart += 100;

}

return CFixedPointNumber(substractIntPart, substractFractPart);

}

CFixedPointNumber CFixedPointNumber::findSubstraction(int value) const {

int substractIntPart = this->intPart - value;

int substractFractPart = this->fractPart;

return CFixedPointNumber(substractIntPart, substractFractPart);

}

CFixedPointNumber CFixedPointNumber::findMultiplication(const CFixedPointNumber& value) const {

double a = ((double)(this->intPart) + (double)(this->fractPart) / 100);

double b = ((double)(value.intPart) + (double)(value.fractPart) / 100);

double multiply = a \* b;

int multiplyIntPart = (int)multiply;

int multiplyFractPart = (int)(multiply - multiplyIntPart) \* 100;

return CFixedPointNumber(multiplyIntPart, multiplyFractPart);

}

CFixedPointNumber CFixedPointNumber::findMultiplication(int value) const {

double a = ((double)(this->intPart) + (double)(this->fractPart) / 100);

int b = value;

double multiply = a \* b;

int multiplyIntPart = (int)multiply;

int multiplyFractPart = (int)(multiply - multiplyIntPart) \* 100;

return CFixedPointNumber(multiplyIntPart, multiplyFractPart);

}

CFixedPointNumber CFixedPointNumber::findDivision(const CFixedPointNumber& value) const {

double a = ((double)(this->intPart) + (double)(this->fractPart) / 100);

double b = ((double)(value.intPart) + (double)(value.fractPart) / 100);

int divisionIntPart = 0;

int divisionFractpart = 0;

if (b != 0) {

double division = a / b;

divisionIntPart = (int)division;

divisionFractpart = (int)((division - divisionIntPart) \* 100);

}

else {

divisionIntPart = 0;

divisionFractpart = 0;

}

return CFixedPointNumber(divisionIntPart, divisionFractpart);

}

CFixedPointNumber CFixedPointNumber::findDivision(int value) const {

double a = ((double)(this->intPart) + (double)(this->fractPart) / 100);

int b = value;

int divisionIntPart = 0;

int divisionFractpart = 0;

if (b != 0) {

double division = a / b;

divisionIntPart = (int)division;

divisionFractpart = (int)((division - divisionIntPart) \* 100);

}

else {

divisionIntPart = 0;

divisionFractpart = 0;

}

return CFixedPointNumber(divisionIntPart, divisionFractpart);

}

CFixedPointNumber CFixedPointNumber::operator+(const CFixedPointNumber& value) const {

int sumIntPart = this->intPart + value.intPart;

int sumFractPart = this->fractPart + value.fractPart;

if (sumFractPart >= 100) {

sumIntPart++;

sumFractPart -= 100;

}

return CFixedPointNumber(sumIntPart, sumFractPart);

}

CFixedPointNumber CFixedPointNumber::operator-(const CFixedPointNumber& value) const {

int substractIntPart = this->intPart - value.intPart;

int substractFractPart = this->fractPart - value.fractPart;

if (substractFractPart < 100) {

substractIntPart--;

substractFractPart += 100;

}

return CFixedPointNumber(substractIntPart, substractFractPart);

}

CFixedPointNumber CFixedPointNumber::operator\*(const CFixedPointNumber& value) const {

double a = ((double)(this->intPart) + (double)(this->fractPart) / 100);

double b = ((double)(value.intPart) + (double)(value.fractPart) / 100);

double multiply = a \* b;

int multiplyIntPart = (int)multiply;

int multiplyFractPart = (int)((multiply - multiplyIntPart) \* 100);

return CFixedPointNumber(multiplyIntPart, multiplyFractPart);

}

CFixedPointNumber CFixedPointNumber::operator/(const CFixedPointNumber& value) const {

double a = ((double)(this->intPart) + (double)(this->fractPart) / 100);

double b = ((double)(value.intPart) + (double)(value.fractPart) / 100);

int divisionIntPart = 0;

int divisionFractpart = 0;

if (b != 0) {

double division = a / b;

divisionIntPart = (int)division;

divisionFractpart = (int)((division - divisionIntPart) \* 100);

}

else {

divisionIntPart = 0;

divisionFractpart = 0;

}

return CFixedPointNumber(divisionIntPart, divisionFractpart);

}

CFixedPointNumber::operator double() const {

return ((double)intPart + ((double)fractPart / 100));

}

std::istream& operator>> (std::istream& input, CFixedPointNumber value) {

input >> value.intPart >> value.fractPart;

return input;

}

std::ostream& operator<< (std::ostream& output, CFixedPointNumber value) {

output << value.getDoubleValue();

return output;

}

bool operator> (const CFixedPointNumber& value, const CFixedPointNumber& otherValue) {

if (value.intPart > otherValue.intPart) {

return true;

}

else if (value.intPart == otherValue.intPart && value.fractPart > otherValue.fractPart) {

return true;

}

else {

return false;

}

}

bool operator< (const CFixedPointNumber& value, const CFixedPointNumber& otherValue) {

if (value.intPart < otherValue.intPart) {

return true;

}

else if (value.intPart == otherValue.intPart && value.fractPart < otherValue.fractPart) {

return true;

}

else {

return false;

}

}

bool operator== (const CFixedPointNumber& value, const CFixedPointNumber& otherValue) {

if (value.intPart == otherValue.intPart && value.fractPart == otherValue.fractPart) {

return true;

}

else {

return false;

}

}

Вміст файлу “main.cpp”:

#include "CMultiSet.h"

#include "CFixedPointNumber.h"

using namespace std;

template <typename T>

int FindElementInArray(T\* array, int size, T element) {

for (int i = 0; i < size; i++) {

if (array[i] == element) {

return i;

}

}

return size;

}

int main() {

CMultiSet<CFixedPointNumber> classMultiset;

CFixedPointNumber value1(2, 45);

CFixedPointNumber value2(2, 44);

CFixedPointNumber value3(5, 80);

CFixedPointNumber value4(4, 00);

classMultiset.addElement(value1);

classMultiset.addElement(value2);

classMultiset.addElement(value3);

classMultiset.addElement(value4);

cout << "Output class multiset: " << classMultiset;

cout << "Min: " << classMultiset.findMin() << endl;

cout << "Sum: " << classMultiset.findSum() << endl;

//cout << "Certain element: " << classMultiset.countOfCertainElement() << endl;

}

**Результат:**

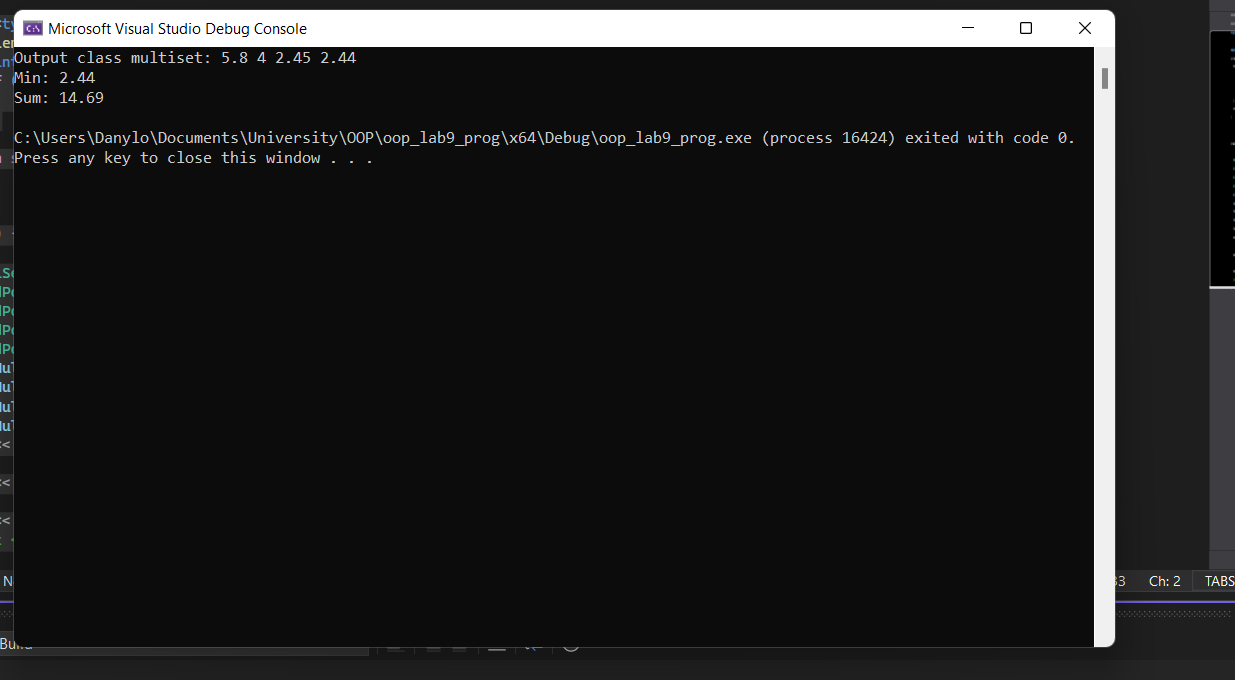
****

Рис. 1

**Висновки**

Отже, в ході виконання лабораторної роботи №9 навчився створювати шаблони функцій та класів, використовувати різні види параметрів шаблонів, а також створювати екземпляри шаблонів.