Лабораторна робота №2

на тему: „Оператори та дружні функції C++“

**Виконав:**

студент II курсу, групи ІП-71

Мокроусов В. Р.

**Перевірив:**

асистент Пономаренко М. Н.

**ЗМІСТ**

1. Мета роботи…………………………………………………………………..3
2. Постановка задачі…………………………………………………………….4
3. Алгоритм вирішення задачі………………………………………………….5
4. UML діаграма………………………………………………………………....6
5. Код програми………………………………………………………………….7
6. Приклад роботи програми…………………………………………………..19
7. Висновок……………………………………………………………………..20

**МЕТА РОБОТИ**

Метою роботи є вивчення особливостей операторних та дружніх функцій, засвоєння особливостей реалізації цих функцій, реалізації операторів для перетворення типів обʼєктів та особливості їх перевантажень.

**ПОСТАНОВКА ЗАДАЧІ**

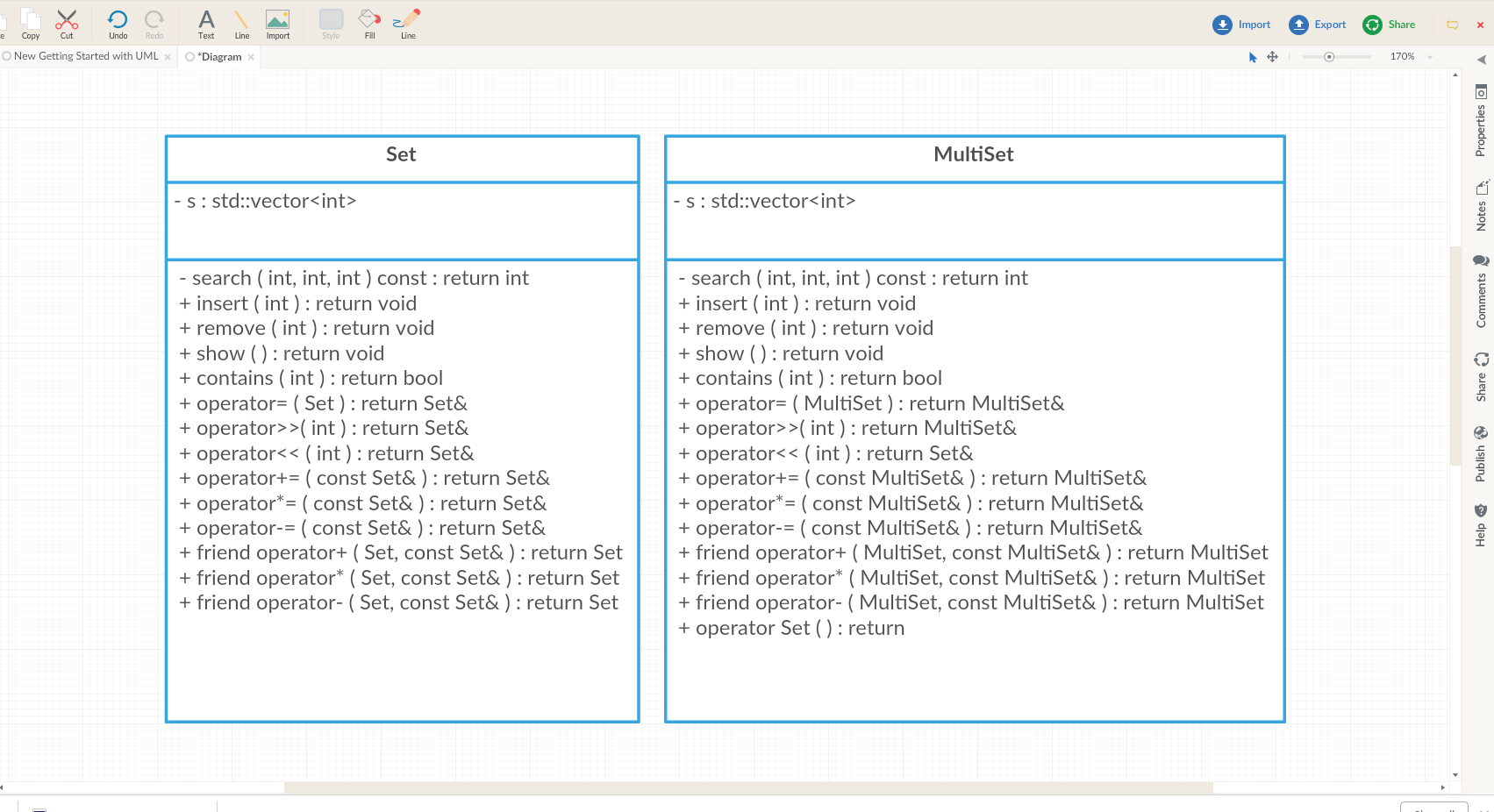
Задачею лабораторної роботи є реалізація класів *Set та MultiSet.* Клас *Set* містить множину упорядкованих унікальних цілочисельних елементів, *MultiSet* містить множину упорядкованих цілочисельних елементів. Для кожного з класів реалізувати оператори об'єднання, перерізу, різниці та оператори +=, -=, \*=. Для класу *MultiSet* реалізувати операцію приведення до класу *Set*. Продемонструвати роботу операцій.

**АЛГОРИТМ ВИРІШЕННЯ ЗАДАЧІ**

Для вирішення задачі було імплементовано класи *Set* та *MultiSet*. Основою для їх реалізації є контейнер *std::*vector. Вміст контейнера підтримується у відсортованому вигляді, для пошуку елемента було використано алгоритм бінарного пошуку. Були також реалізовані методи вставки, видалення, перевірки на вміст елемента, оператори перерізу, обʼєднання, різниці та оператори додавання та видалення елементу зі структури.

**UML ДІАГРАМА**

Нижче наведено UML діаграму класу *Set* та *MultiSet*.



**КОД ПРОГРАМИ**

// main.cpp

#include "Set.hpp"

#include "MultiSet.hpp"

#include <iostream>

#include <string>

#include <time.h>

#define N 5

using namespace std;

void show(string, Set&, bool);

void show(string, MultiSet&, bool);

int main(int argc, char\* argv[]) {

MultiSet mset1, mset2, unite, intersection, diff;

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

mset1 << rand()%N;

mset2 << rand()%N;

}

// operations

unite = mset1 + mset2;

intersection = mset1 \* mset2;

diff = mset1 - mset2;

cout << "Creating MultiSets" << endl;

show("MultiSet 1", mset1, true);

show("MultiSet 2", mset2, true);

cout << " ------- " << endl;

cout << "Operation : Unite" << endl << "Operands : ";

show("", mset1, false);

show("", mset2, false);

cout << endl;

show("Unite", unite, true);

cout << " ------- " << endl;

cout << "Operation : Intersection" << endl << "Operands : ";

show("", mset1, false);

show("", mset2, false);

cout << endl;

show("Intersection", intersection, true);

cout << " ------- " << endl;

cout << "Operation : Diff" << endl << "Operands : ";

show("", mset1, false);

show("", mset2, false);

cout << endl;

show("Diff", diff, true);

cout << " ------- " << endl;

cout << "Operation : Add assignement" << endl << "Operands : ";

show("", mset1, false);

show("", mset2, false);

mset1 += mset2;

cout << endl;

show("MultiSet 1", mset1, true);

cout << " ------- " << endl;

cout << "Operation : Intersection assignement" << endl << "Operands : ";

show("", mset1, false);

show("", mset2, false);

mset1 \*= mset2;

cout << endl;

show("MultiSet 1", mset1, true);

cout << " ------- " << endl;

cout << "Operation : Diff assignement" << endl << "Operands : ";

show("", mset1, false);

show("", mset2, false);

mset1 -= mset2;

cout << endl;

show("MultiSet 1", mset1, true);

// delete/add op

cout << " ------- " << endl;

cout << "Operation : Add" << endl << "Operands : ";

show("", mset1, false);

cout << " { 0 1 2 } " << endl;

mset1 << 0 << 1 << 2;

show("MultiSet 1", mset1, true);

cout << " ------- " << endl;

cout << "Operation : Delete" << endl << "Operands : ";

show("", mset2, false);

cout << " { 0 1 2 } " << endl;

mset2 >> 0 >> 1 >> 2;

show("MultiSet 2", mset2, true);

// Add repeating numbers

cout << " ------- " << endl;

cout << "Operation : Add repeating " << endl << "Operands : ";

show("", mset1, false);

cout << " { 100 100 100 } " << endl;

mset1 << 100 << 100 << 100;

show("MultiSet 1", mset1, true);

cout << " ------- " << endl;

cout << "Operation : Add repeating " << endl << "Operands : ";

show("", mset2, false);

cout << " { 99 99 99 } " << endl;

mset2 << 99 << 99 << 99;

show("MultiSet 2", mset2, true);

// Conversion operatorx

cout << " ------- " << endl;

cout << "Operation : Set conversion " << endl << "Operands : " << endl;

show("MultiSet 1", mset1, true);

show("MultiSet 2", mset2, true);

Set set1 = Set(mset1);

Set set2 = Set(mset2);

cout << "Result : " << endl;

show("Set 1", set1, true);

show("Set 2", set2, true);

}

void show(string message, Set& set, bool flag = false) {

cout << message << " { ";

set.show();

cout << "} ";

if (flag) cout << endl;

}

void show(string message, MultiSet& set, bool flag = false) {

cout << message << " { ";

set.show();

cout << "} ";

if (flag) cout << endl;

}

// Set.hpp

#ifndef SET\_HPP

#define SET\_HPP

#include <iostream>

#include <vector>

#include <utility>

class Set {

std::vector<int> s;

int search(int lo, int hi, int key) const {

if (hi < lo) return -1;

int mid = (lo + hi) / 2;

if (key == s[mid]) return mid;

if (key > s[mid]) return search(mid + 1, hi, key);

return search(lo, mid - 1, key);

}

public:

// main functions

void insert(int key) {

if (contains(key)) {

return;

}

if (s.size() == 0) {

s.push\_back(key);

return;

}

int i, n = s.size();

s.push\_back(key);

for (i = n - 1; i >= 0 && s[i] >= key; i--) s[i + 1] = s[i];

s[i + 1] = key;

}

void remove(int key) {

int index = search(0, s.size(), key);

if (index == -1) return;

s.erase(s.begin() + index);

}

void show() const { for (int key : s) std::cout << key << " "; }

bool contains(int key) const {

if (s.size() == 0) return false;

if (search(0, s.size()-1, key) == -1) return false;

return true;

}

// operators

Set& operator=(Set that) {

this->s.swap(that.s);

return \*this;

}

Set& operator>>(int key) {

this->remove(key);

return \*this;

}

Set& operator<<(int key) {

this->insert(key);

return \*this;

}

Set& operator+=(const Set& set) {

Set temp = \*this + set;

this->s.swap(temp.s);

return \*this;

}

Set& operator\*=(const Set& set) {

Set temp = \*this \* set;

this->s.swap(temp.s);

return \*this;

}

Set& operator-=(const Set& set) {

Set temp = \*this - set;

this->s.swap(temp.s);

return \*this;

}

friend Set operator+(Set a, const Set& b) {

std::vector<int> it = b.s;

for (int key : it) a.insert(key);

return a;

}

friend Set operator\*(Set a, const Set& b) {

Set result;

std::vector<int> it = b.s;

for (int key : it) if (a.contains(key)) result.insert(key);

return result;

}

friend Set operator-(Set a, const Set& b) {

Set result;

std::vector<int> it = a.s;

for (int key : it) if (!b.contains(key)) result.insert(key);

return result;

}

};

#endif

// MultiSet.hpp

#ifndef MULTISET\_HPP

#define MULTISET\_HPP

#include "Set.hpp"

#include <iostream>

#include <vector>

#include <utility>

class MultiSet {

std::vector<int> s;

int search(int low, int high, int key) const {

if (high < low) return -1;

int mid = (low + high) / 2;

if (key == s[mid]) return mid;

if (key > s[mid]) return search(mid + 1, high, key);

return search(low, mid - 1, key);

}

public:

// main functions

void insert(int key) {

if (s.size() == 0) {

s.push\_back(key);

return;

}

int i, n = s.size();

s.push\_back(key);

for (i = n - 1; i >= 0 && s[i] >= key; i--) {

s[i + 1] = s[i];

}

s[i + 1] = key;

}

//

void remove(int key) {

int index = search(0, s.size(), key);

if (index == -1) return;

s.erase(s.begin() + index);

}

void show() const { for (int key : s) std::cout << key << " "; }

bool contains(int key) const {

if (s.size() == 0) return false;

if (search(0, s.size(), key) == -1) return false;

return true;

}

// operators

MultiSet& operator=(MultiSet that) {

this->s.swap(that.s);

return \*this;

}

MultiSet& operator>>(int key) {

this->remove(key);

return \*this;

}

MultiSet& operator<<(int key) {

this->insert(key);

return \*this;

}

MultiSet& operator+=(const MultiSet& set) {

MultiSet temp = \*this + set;

this->s.swap(temp.s);

return \*this;

}

MultiSet& operator\*=(const MultiSet& set) {

MultiSet temp = \*this \* set;

this->s.swap(temp.s);

return \*this;

}

MultiSet& operator-=(const MultiSet& set) {

MultiSet temp = \*this - set;

this->s.swap(temp.s);

return \*this;

}

friend MultiSet operator+(MultiSet a, const MultiSet& b) {

std::vector<int> it = b.s;

for (int key : it) a.insert(key);

return a;

}

friend MultiSet operator\*(MultiSet a, const MultiSet& b) {

MultiSet result;

std::vector<int> it = b.s;

for (int key : it) if (a.contains(key)) result.insert(key);

return result;

}

friend MultiSet operator-(MultiSet a, const MultiSet& b) {

MultiSet result;

std::vector<int> it = a.s;

for (int key : it) if (!b.contains(key)) result.insert(key);

return result;

}

operator Set() {

Set result;

for (int key : this->s) result.insert(key);

return result;

}

};

#endif

//Makefile

CPPFLAGS=-Wpedantic -std=c++14 -g

OFNAME=main

SRC=main.cpp

build:

@g++ ${CPPFLAGS} ${SRC} -o ${OFNAME}

run: build

@./${OFNAME}

**ПРИКЛАД РОБОТИ ПРОГРАМИ**

Программа компілюється командою *make* (за допомогою *Makefile*), та запускається в консолі *./main*.

****

**ВИСНОВОК**

При виконанні лабораторної роботи було, згідно умови, створено відповідні класи. Для них було реалізовано усі визначені умовою методи та оператори. При виконанні було використано механізми дружніх функцій та перевантаження операторів. Було використано елементи стандартної бібліотеки C++, зокрема функції для виводу результату, контейнер *std::vector* та інші.

В результаті було створено ефективну програму для розв'язку та поставленої задачі, згідно всіх поставлених умов.