Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский

политехнический университет»

Лабораторная работа №13

«Стандартные обобщенные алгоритмы библиотеки STL»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-3б

Коротаев Александр Дмитриевич

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

**Цель задания**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Использование стандартных обобщенных алгоритмов из библиотеки STL в ОО программе.

**Постановка задачи**

Задача 1.

1. Создать последовательный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Заменить элементы в соответствии с заданием (использовать

алгоритмы

replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

1. Удалить элементы в соответствии с заданием (использовать

алгоритмы

remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

1. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
2. Найти в контейнере заданный элемент (использовать алгоритмы

find(), find\_if(), count(), count\_if()).

1. Выполнить задание варианта для полученного контейнера

(использовать алгоритм for\_each()) .

1. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 2.

1. Создать адаптер контейнера.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Заменить элементы в соответствии с заданием (использовать

алгоритмы

replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

1. Удалить элементы в соответствии с заданием (использовать

алгоритмы

remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

1. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
2. Найти в контейнере элемент с заданным ключевым полем

(использовать алгоритмы find(), find\_if(), count(), count\_if()).

1. Выполнить задание варианта для полученного контейнера

(использовать алгоритм for\_each()) .

1. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 3

1. Создать ассоциативный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Заменить элементы в соответствии с заданием (использовать

алгоритмы

replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

1. Удалить элементы в соответствии с заданием (использовать

алгоритмы

remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

1. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
2. Найти в контейнере элемент с заданным ключевым полем

(использовать алгоритмы find(), find\_if(), count(), count\_if()).

1. Выполнить задание варианта для полученного контейнера

(использовать алгоритм for\_each()) .

1. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

**Анализ задачи 1**

1. Необходимо реализовать класс Pair. В классе надо прописать конструкторы: по умолчанию, с параметрами и копирования; а также деструктор, для предотвращения утечки памяти.
2. Перегруженные операторы +, -, = и << позволяют выполнять

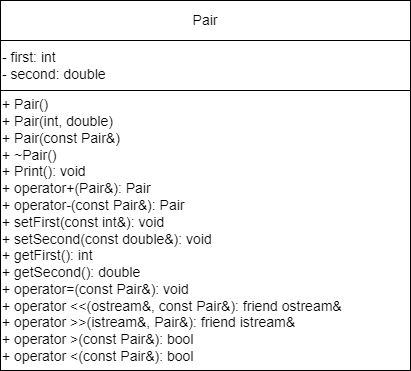
различные операции с объектами класса Pair. Например, оператор + позволяет складывать целое число с первым полем объекта класса и вещественное число со вторым полем соответственно. Оператор << позволяет выводить объекты класса Pair в поток вывода.

1. Для использования списка необходимо подключить библиотеку

#include <list>.

1. При создании списка необходимо указать какие объекты будут в нем храниться. В данном случае – объекты класса Pair: list<Pair> list\_1(N)
2. Для использования лямда функций и других необходимо подключить: #include <functional>, #include <algorithm>.
3. При поиске максимального объекта будем считать, что необходимо сравнивать сначала по первому числу, а затем по второму числу в паре. Аналогично для поиска минимального элемента, и сортировки списка.
4. В главной функции показаны операции, которые можно выполнить с контейнером.

**UML диаграмма**



**Код**

Файл ООП13.1.cpp

#include <iostream>

#include <list>

#include <functional>

#include <algorithm>

#include "Pair.h"

using namespace std;

ostream& operator<<(ostream&, list<Pair>&);

void randomize(Pair&);

int main()

{

system("chcp 1251 > Null");

srand(time(0));

int list\_size;

do

{

cout << "Введите размер списка: ";

cin >> list\_size;

} while (list\_size < 1);

Pair sum;

list<Pair> list\_1(list\_size);

generate(list\_1.begin(), list\_1.end(), []() {Pair p; randomize(p); return p; });

cout << "Сгенерированный список:\n" << list\_1 << "\n";

for\_each(list\_1.begin(), list\_1.end(), [&sum](Pair& p) {sum = sum + p; });

sum.setFirst(sum.getFirst() / list\_1.size());

sum.setSecond(sum.getSecond() / list\_1.size());

cout << "Среднее арифметическое " << sum << "\n\n";

list\_1.push\_back(sum);

cout << "После добавления среднего арифметического:\n" << list\_1 << "\n";

Pair first\_el, second\_el;

cout << "Введите две пары чисел, из диапазона которых будут удалены пары списка\n";

cout << "Первая пара:\n";

cin >> first\_el;

cout << "Вторая пара:\n";

cin >> second\_el;

if (first\_el > second\_el)

{

swap(first\_el, second\_el);

}

list<Pair>::iterator iter = remove\_if(list\_1.begin(), list\_1.end(),

[first\_el, second\_el](Pair& p) {return p > first\_el && p < second\_el; });

list\_1.erase(iter, list\_1.end());

if (list\_1.size() != 0)

{

cout << "\nПосле удаления из диапазона\n" << list\_1 << "\n";

pair<list<Pair>::iterator, list<Pair>::iterator> minmax\_pair = minmax\_element(list\_1.begin(), list\_1.end());

cout << "Минимальный элемент: " << \*minmax\_pair.first << "\n";

cout << "Максимальный элемент: " << \*minmax\_pair.second << "\n\n";

Pair sum\_pair(minmax\_pair.first->getFirst() + minmax\_pair.second->getFirst(),

minmax\_pair.first->getSecond() + minmax\_pair.second->getSecond());

cout << "Сумма максимального и минимального элементов " << sum\_pair << "\n";

for\_each(list\_1.begin(), list\_1.end(), [&sum\_pair](Pair& p) {p = p + sum\_pair; });

cout << "После добавления суммы максимального и минимального к каждой паре чисел:\n";

cout << list\_1 << "\n";

cout << "Сортировка по Возрастанию:\n";

list\_1.sort();

cout << list\_1 << "\n";

cout << "Сортировка по Убыванию:\n";

list\_1.sort([](Pair& a, Pair& b) {return a > b; });

cout << list\_1 << "\n";

cout << "Введите значение first, по которому необходимо найти пару: ";

int temp\_first;

cin >> temp\_first;

auto i = find\_if(list\_1.begin(), list\_1.end(), [&temp\_first](Pair& p) {return temp\_first == p.getFirst(); });

if (i != list\_1.end())

{

cout << "Элемент найден: " << \*i << "\n";

}

else

{

cout << "Элемент не найден =(\n";

}

}

else

{

cout << "\nСписок пуст\n";

}

return 0;

}

ostream& operator<<(ostream& stream, list<Pair>& l)

{

for\_each(l.begin(), l.end(), [](Pair& p) {cout << p << "\n"; });

return stream;

}

void randomize(Pair& p)

{

int x = rand() % 1001;

double y = (rand() % 100001) \* 0.01;

p = Pair(x, y);

}

Файл Pair.h

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

Pair();

Pair(int, double);

Pair(const Pair&);

~Pair() {}

void Print();

Pair operator+(const Pair&);

Pair operator-(const Pair&);

void setFirst(const int&);

void setSecond(const double&);

int getFirst();

double getSecond();

void operator=(const Pair&);

friend ostream& operator <<(ostream&, const Pair&);

friend istream& operator >>(istream&, Pair&);

bool operator >(const Pair&);

bool operator <(const Pair&);

};

Файл Pair.cpp

#pragma once

#include "Pair.h"

#include <iostream>

#include <string>

using namespace std;

Pair::Pair()

{

this->first = 0;

this->second = 0;

}

Pair::Pair(int first, double second)

{

this->first = first;

this->second = second;

}

Pair::Pair(const Pair& other)

{

this->first = other.first;

this->second = other.second;

}

void Pair::Print()

{

cout << "(" << this->first << " : " << this->second << ")\n";

}

Pair Pair::operator+(const Pair& p)

{

return (Pair(this->first + p.first, this->second + p.second));

}

Pair Pair::operator-(const Pair& p)

{

return (Pair(this->first - p.first, this->second - p.second));

}

void Pair::setFirst(const int& x)

{

this->first = x;

}

void Pair::setSecond(const double& y)

{

this->second = y;

}

int Pair::getFirst()

{

return this->first;

}

double Pair::getSecond()

{

return this->second;

}

void Pair::operator=(const Pair& p)

{

this->first = p.first;

this->second = p.second;

}

ostream& operator <<(ostream& stream, const Pair& p)

{

stream << "(" << p.first << " : " << p.second << ")";

return stream;

}

istream& operator >>(istream& stream, Pair& p)

{

cout << "Введите число first и second через пробел: ";

stream >> p.first >> p.second;

return stream;

}

bool Pair::operator >(const Pair& p)

{

if (this->first >= p.first && this->second > p.second)

{

return true;

}

if (this->first > p.first && this->second >= p.second)

{

return true;

}

return false;

}

bool Pair::operator <(const Pair& p)

{

if (this->first <= p.first && this->second < p.second)

{

return true;

}

if (this->first < p.first && this->second <= p.second)

{

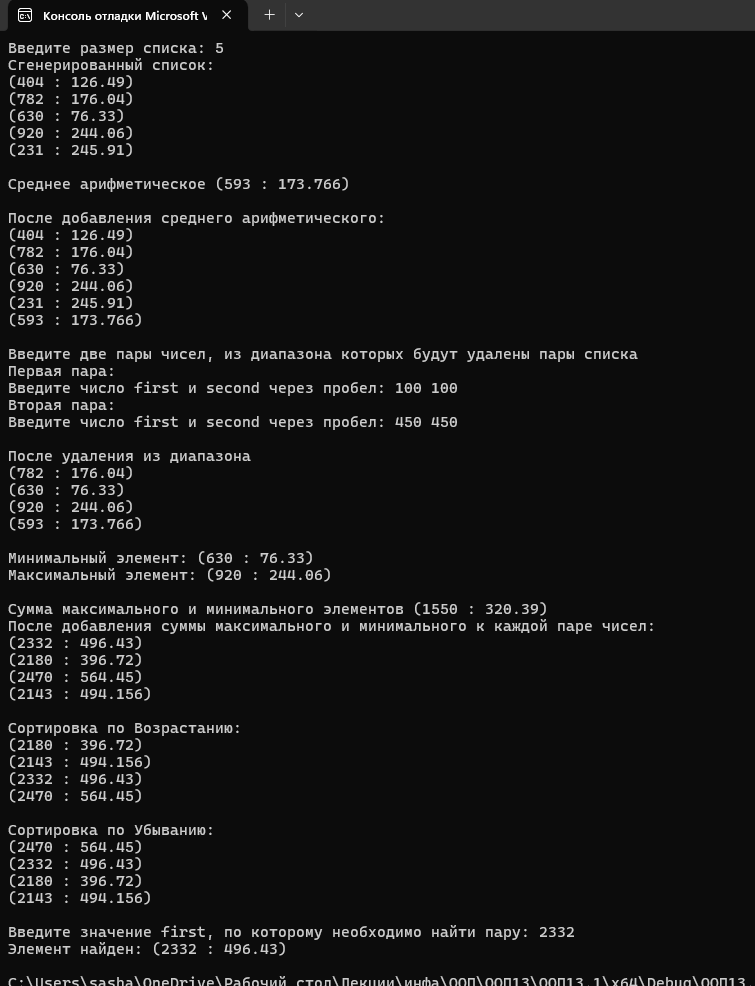
return true;

}

return false;

}

**Результаты работы**

****

**Анализ задачи 2**

1. Необходимо реализовать класс Pair. В классе надо прописать конструкторы: по умолчанию, с параметрами и копирования; а также деструктор, для предотвращения утечки памяти.
2. Перегруженные операторы +, -, = и << позволяют выполнять

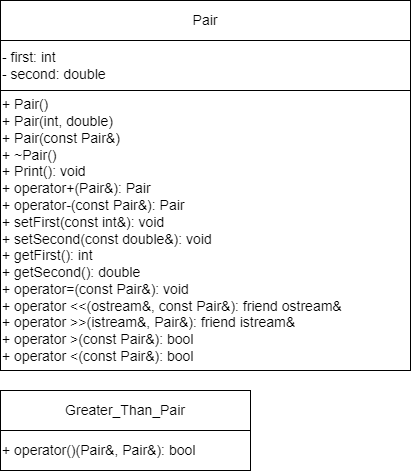
различные операции с объектами класса Pair. Например, оператор + позволяет складывать целое число с первым полем объекта класса и вещественное число со вторым полем соответственно. Оператор << позволяет выводить объекты класса Pair в поток вывода.

1. Для использования очереди необходимо подключить библиотеку

#include < queue>.

1. При создании очереди с приоритетом необходимо указать какие объекты будут в нем храниться. В данном случае – объекты класса Pair: priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> qst.
2. Для использования лямда функций и других необходимо подключить: #include <functional>, #include <algorithm>.
3. При поиске максимального объекта будем считать, что необходимо сравнивать сначала по первому числу, а затем по второму числу в паре.
4. В главной функции показаны операции, которые можно выполнить с контейнером.

**UML диаграмма**

****

**Код**

Файл ООП13.2.cpp

#include <iostream>

#include <queue>

#include <functional>

#include <algorithm>

#include "Pair.h"

#include <random>

using namespace std;

class Greater\_Than\_Pair

{

public:

bool operator()(Pair& p1, Pair& p2)

{

return p1 < p2;

}

};

void randomize(Pair&);

void generate\_Q(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>&, int);

ostream& operator<<(ostream&, priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>);

Pair all\_sum(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>);

void remove\_from\_range(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>&, Pair&, Pair&);

Pair maximum\_pair(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>&);

Pair minimum\_pair(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>);

priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> add\_to\_all(priority\_queue<Pair,

vector<Pair>, Greater\_Than\_Pair>&, Pair&);

bool find\_para(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>, int&, Pair&);

int main()

{

system("chcp 1251 > Null");

srand(time(0));

int N;

do

{

cout << "Введите размер очереди: ";

cin >> N;

} while (N < 1);

priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> qst;

generate\_Q(qst, N);

cout << qst;

Pair sum = all\_sum(qst);

sum.setFirst(sum.getFirst() / qst.size());

sum.setSecond(sum.getSecond() / qst.size());

cout << "\nСреднее арифметическое: " << sum << "\n\n";

qst.push(sum);

cout << "После добавления среднего арифметического:\n" << qst << "\n";

Pair first\_el, second\_el;

cout << "Введите две пары чисел, из диапазона которых будут удалены пары списка\n";

cout << "Первая пара\n"; cin >> first\_el;

cout << "Вторая пара\n"; cin >> second\_el;

if (first\_el > second\_el)

{

swap(first\_el, second\_el);

}

remove\_from\_range(qst, first\_el, second\_el);

cout << "\nПосле удаления из диапазона:\n" << qst << "\n";

Pair max\_el = maximum\_pair(qst);

Pair min\_el = minimum\_pair(qst);

cout << "Минимальный элемент: " << min\_el << "\n";

cout << "Максимальный элемент: " << max\_el << "\n\n";

sum = max\_el + min\_el;

cout << "Сумма максимального и минимального элементов: " << sum << "\n";

qst = add\_to\_all(qst, sum);

cout << "После добавления суммы максимального и минимального к каждой паре чисел:\n" << qst;

cout << "Введите значение first, по которому необходимо найти пару: ";

int tmp\_first;

cin >> tmp\_first;

Pair finded\_element;

bool flag = find\_para(qst, tmp\_first, finded\_element);

if (flag)

{

cout << "Элемент найден: " << finded\_element << "\n";

}

else

{

cout << "Элемент не найден =(\n";

}

return 0;

}

void randomize(Pair& p)

{

int x = rand() % 1001;

double y = (rand() % 100001) \* 0.01;

p = Pair(x, y);

}

void generate\_Q(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>& my\_Q, int Q\_size)

{

Pair tmp(0, 0);

for (int i = 0; i < Q\_size; i++)

{

randomize(tmp);

my\_Q.push(tmp);

}

}

ostream& operator<<(ostream& stream, priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> my\_Q)

{

while (!my\_Q.empty())

{

Pair user = my\_Q.top();

cout << user << endl;

my\_Q.pop();

}

return stream;

}

Pair all\_sum(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> temp\_Q)

{

Pair sum(0, 0);

while (!temp\_Q.empty())

{

Pair top\_pair = temp\_Q.top();

sum = sum + top\_pair;

temp\_Q.pop();

}

return sum;

}

void remove\_from\_range(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>& my\_Q, Pair& range\_1, Pair& range\_2)

{

priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> queue\_1;

while (!my\_Q.empty())

{

Pair top\_pair = my\_Q.top();

if (!(top\_pair > range\_1 && top\_pair < range\_2))

{

queue\_1.push(top\_pair);

}

my\_Q.pop();

}

my\_Q = queue\_1;

}

Pair maximum\_pair(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair>& my\_Q)

{

return my\_Q.top();

}

Pair minimum\_pair(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> my\_Q)

{

Pair min\_p;

while (!my\_Q.empty())

{

min\_p = my\_Q.top();

my\_Q.pop();

}

return min\_p;

}

priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> add\_to\_all(priority\_queue<Pair,

vector<Pair>, Greater\_Than\_Pair>& my\_Q, Pair& pair)

{

priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> queue\_1;

Pair top\_pair = my\_Q.top();

while (!my\_Q.empty())

{

top\_pair = my\_Q.top();

queue\_1.push(top\_pair + pair);

my\_Q.pop();

}

return queue\_1;

}

bool find\_para(priority\_queue<Pair, vector<Pair>, Greater\_Than\_Pair> my\_Q, int& x, Pair& finded\_element)

{

bool flag = false;

while (!my\_Q.empty() && !flag)

{

finded\_element = my\_Q.top();

if (x == finded\_element.getFirst())

{

flag = true;

}

my\_Q.pop();

}

return flag;

}

Файл Pair.h

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

Pair();

Pair(int, double);

Pair(const Pair&);

~Pair() {}

void Print();

Pair operator+(const Pair&);

Pair operator-(const Pair&);

void setFirst(const int&);

void setSecond(const double&);

int getFirst();

double getSecond();

void operator=(const Pair&);

friend ostream& operator <<(ostream&, const Pair&);

friend istream& operator >>(istream&, Pair&);

bool operator >(const Pair&);

bool operator <(const Pair&);

};

Файл Pair.cpp

#pragma once

#include "Pair.h"

#include <iostream>

#include <string>

using namespace std;

Pair::Pair()

{

this->first = 0;

this->second = 0;

}

Pair::Pair(int first, double second)

{

this->first = first;

this->second = second;

}

Pair::Pair(const Pair& other)

{

this->first = other.first;

this->second = other.second;

}

void Pair::Print()

{

cout << "(" << this->first << " : " << this->second << ")\n";

}

Pair Pair::operator+(const Pair& p)

{

return (Pair(this->first + p.first, this->second + p.second));

}

Pair Pair::operator-(const Pair& p)

{

return (Pair(this->first - p.first, this->second - p.second));

}

void Pair::setFirst(const int& x)

{

this->first = x;

}

void Pair::setSecond(const double& y)

{

this->second = y;

}

int Pair::getFirst()

{

return this->first;

}

double Pair::getSecond()

{

return this->second;

}

void Pair::operator=(const Pair& p)

{

this->first = p.first;

this->second = p.second;

}

ostream& operator <<(ostream& stream, const Pair& p)

{

stream << "(" << p.first << " : " << p.second << ")";

return stream;

}

istream& operator >>(istream& stream, Pair& p)

{

cout << "Введите число first и second через пробел: ";

stream >> p.first >> p.second;

return stream;

}

bool Pair::operator >(const Pair& p)

{

if (this->first >= p.first && this->second > p.second)

{

return true;

}

if (this->first > p.first && this->second >= p.second)

{

return true;

}

return false;

}

bool Pair::operator <(const Pair& p)

{

if (this->first <= p.first && this->second < p.second)

{

return true;

}

if (this->first < p.first && this->second <= p.second)

{

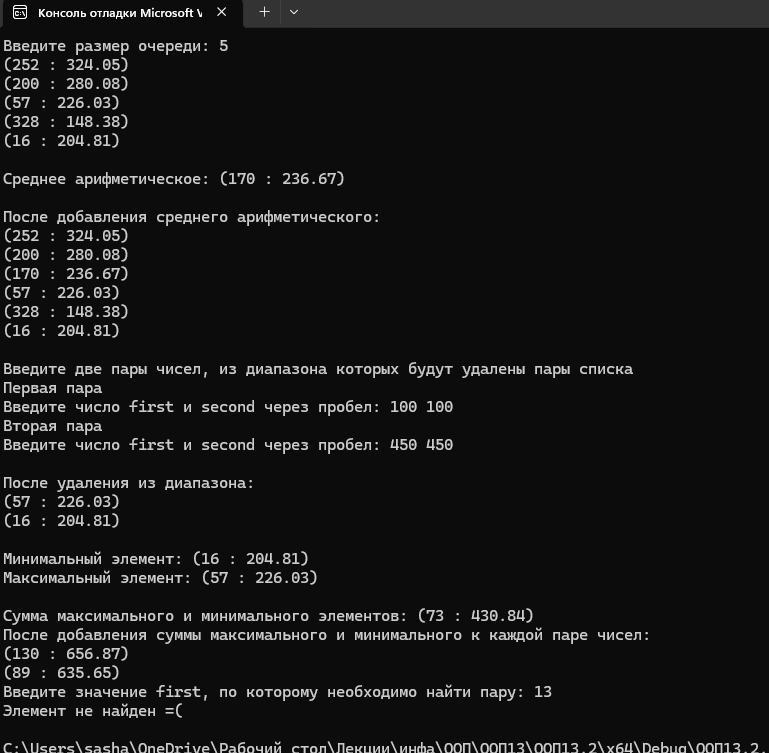
return true;

}

return false;

}

**Результаты работы**



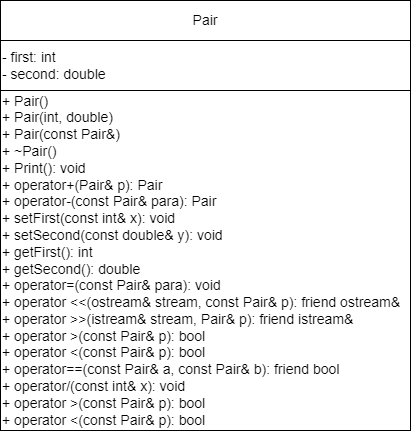
**Анализ задачи 3**

1. Необходимо реализовать класс Pair. В классе надо прописать конструкторы: по умолчанию, с параметрами и копирования; а также деструктор, для предотвращения утечки памяти.
2. Перегруженные операторы +, -, = и << позволяют выполнять различные операции с объектами класса Pair. Например, оператор + позволяет складывать целое число с первым полем объекта класса и вещественное число со вторым полем соответственно. Оператор << позволяет выводить объекты класса Pair в поток вывода.
3. Для использования словаря необходимо подключить библиотеку

#include <map>.

1. При создании очереди с приоритетом необходимо указать какие объекты будут в нем храниться. В данном случае – объекты класса Pair: map<Pair, Pair>.
2. Для использования лямда функций и других необходимо подключить: #include <functional>, #include <algorithm>.
3. При поиске максимального объекта будем считать, что необходимо сравнивать сначала по первому числу, а затем по второму числу в паре.
4. В главной функции показаны операции, которые можно выполнить с контейнером.

**UML диаграмма**



**Код**

Файл ООП13.3.cpp

#include <iostream>

#include <functional>

#include <algorithm>

#include "Pair.h"

#include <map>

using namespace std;

void randomize(Pair&);

void generate\_M(map<Pair, Pair>&, int);

ostream& operator<<(ostream&, const pair<Pair, Pair>);

ostream& operator<<(ostream&, const map<Pair, Pair>&);

bool isInRange(Pair&, Pair&, Pair&);

void remove\_from\_range(map<Pair, Pair>&, Pair&, Pair&);

Pair minimum\_pair(map<Pair, Pair>&);

Pair maximum\_pair(map<Pair, Pair>&);

void all\_sum(map<Pair, Pair>&, Pair&, Pair&);

map<Pair, Pair> add\_to\_all(map<Pair, Pair>&, Pair&, Pair&);

bool findPair(map<Pair, Pair>&, Pair&, Pair&);

int main()

{

system("chcp 1251 > Null");

srand(time(0));

int N;

do {

cout << "Введите размер словаря: ";

cin >> N;

} while (N < 1);

Pair sum\_1(0, 0), sum\_2(0, 0), sum\_key(0, 0), sum\_value(0, 0);

map<Pair, Pair> mapp;

generate\_M(mapp, N);

cout << "Сгенерированный словарь:\n" << mapp << "\n";

all\_sum(mapp, sum\_1, sum\_2);

sum\_1 / mapp.size();

sum\_2 / mapp.size();

cout << "Среднее арифметическое ключей " << sum\_1 << "\n";

cout << "Среднее арифметическое значений " << sum\_2 << "\n\n";

mapp[sum\_1] = sum\_2;

cout << "После добавления среднего арифметического:\n" << mapp << "\n";

Pair first\_el, second\_el;

cout << "Введите две пары чисел (ключей), из диапазона которых будут удалены пары словаря\n";

cout << "Первая пара:\n";

cin >> first\_el;

cout << "Вторая пара:\n";

cin >> second\_el;

if (first\_el > second\_el)

{

swap(first\_el, second\_el);

}

remove\_from\_range(mapp, first\_el, second\_el);

cout << "\nПосле удаления из диапазона\n" << mapp << "\n";

Pair max\_el = maximum\_pair(mapp);

Pair min\_el = minimum\_pair(mapp);

findPair(mapp, min\_el, sum\_1);

findPair(mapp, max\_el, sum\_2);

cout << "Минимальный элемент по ключу: " << min\_el << "\n";

cout << "Ключ - " << min\_el << " Значение - " << sum\_1 << "\n";

cout << "Максимальный элемент по ключу: " << max\_el << "\n";

cout << "Ключ - " << max\_el << " Значение - " << sum\_2 << "\n\n";

sum\_key = min\_el + max\_el;

sum\_value = sum\_1 + sum\_2;

cout << "Сумма максимального и минимального элементов:\n";

cout << "Ключ - " << sum\_key << " Значение - " << sum\_value << "\n\n";

mapp = add\_to\_all(mapp, sum\_key, sum\_value);

cout << "После добавления суммы максимального и минимального к каждой паре чисел:\n";

cout << mapp << "\n";

cout << "Введите ключевое значение, по которому необходимо найти пару\n";

cin >> first\_el;

bool found = findPair(mapp, first\_el, second\_el);

cout << "\n";

if (found)

{

cout << "Элемент найден.\n";

cout << "Ключ - " << first\_el << " Значение - " << second\_el << "\n";

}

else

{

cout << "Элемент не найден =(\n";

}

return 0;

}

void randomize(Pair& p)

{

int x = rand() % 1001;

double y = (rand() % 100001) \* 0.01;

p = Pair(x, y);

}

void generate\_M(map<Pair, Pair>& my\_M, int M\_size)

{

Pair qwe\_1(0, 0), qwe\_2(0, 0);

for (int i = 0; i < M\_size; i++)

{

randomize(qwe\_1);

randomize(qwe\_2);

my\_M[qwe\_1] = qwe\_2;

}

}

ostream& operator<<(ostream& stream, const pair<Pair, Pair> p)

{

stream << "Ключ - " << p.first << "\tЗначение - " << p.second << "\n";

return stream;

}

ostream& operator<<(ostream& stream, const map<Pair, Pair>& my\_M)

{

map<Pair, Pair>::const\_iterator Iter = my\_M.begin();

while (Iter != my\_M.end())

{

cout << \*Iter++;

}

return stream;

}

bool isInRange(Pair& key, Pair& lower, Pair& upper)

{

return key > lower && key < upper;

}

void remove\_from\_range(map<Pair, Pair>& my\_M, Pair& range\_1, Pair& range\_2)

{

auto iterator = my\_M.begin();

Pair key;

while (iterator != my\_M.end())

{

key = iterator->first;

if (!isInRange(key, range\_1, range\_2))

{

++iterator;

}

else

{

iterator = my\_M.erase(iterator);

}

}

}

Pair minimum\_pair(map<Pair, Pair>& my\_M)

{

return my\_M.begin()->first;

}

Pair maximum\_pair(map<Pair, Pair>& my\_M)

{

auto iter = my\_M.begin();

Pair tmp\_pair;

while (iter != my\_M.end())

{

tmp\_pair = iter->first;

++iter;

}

return tmp\_pair;

}

void all\_sum(map<Pair, Pair>& my\_M, Pair& key, Pair& value)

{

for (const auto& pair : my\_M)

{

Pair tmp = pair.first;

key = key + tmp;

tmp = pair.second;

value = value + tmp;

}

}

map<Pair, Pair> add\_to\_all(map<Pair, Pair>& my\_M, Pair& key, Pair& value)

{

map<Pair, Pair> tmp\_map;

for (const auto& pair : my\_M)

{

Pair tmp\_key = pair.first;

Pair tmp\_value = pair.second;

tmp\_map[tmp\_key + key] = (tmp\_value + value);

}

return tmp\_map;

}

bool findPair(map<Pair, Pair>& my\_M, Pair& key, Pair& value)

{

auto iter = my\_M.find(key);

if (iter != my\_M.end())

{

value = Pair(iter->second);

return true;

}

else

{

return false;

}

}

Файл Pair.h

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

Pair();

Pair(int, double);

Pair(const Pair&);

~Pair() {}

void Print();

Pair operator+(const Pair&);

Pair operator-(const Pair&);

void setFirst(const int&);

void setSecond(const double&);

int getFirst();

double getSecond();

void operator=(const Pair&);

friend ostream& operator <<(ostream&, const Pair&);

friend istream& operator >>(istream&, Pair&);

bool operator >(const Pair&);

bool operator <(const Pair&);

friend bool operator==(const Pair&, const Pair&);

void operator/(const int&);

friend bool operator<(const Pair&, const Pair&);

friend bool operator>(const Pair&, const Pair&);

};

Файл Pair.cpp

#pragma once

#include "Pair.h"

#include <iostream>

#include <string>

using namespace std;

Pair::Pair()

{

this->first = 0;

this->second = 0;

}

Pair::Pair(int first, double second)

{

this->first = first;

this->second = second;

}

Pair::Pair(const Pair& other)

{

this->first = other.first;

this->second = other.second;

}

void Pair::Print()

{

cout << "(" << this->first << " : " << this->second << ")\n";

}

Pair Pair::operator+(const Pair& p)

{

return (Pair(this->first + p.first, this->second + p.second));

}

Pair Pair::operator-(const Pair& p)

{

return (Pair(this->first - p.first, this->second - p.second));

}

void Pair::setFirst(const int& x)

{

this->first = x;

}

void Pair::setSecond(const double& y)

{

this->second = y;

}

int Pair::getFirst()

{

return this->first;

}

double Pair::getSecond()

{

return this->second;

}

void Pair::operator=(const Pair& p)

{

this->first = p.first;

this->second = p.second;

}

ostream& operator <<(ostream& stream, const Pair& p)

{

stream << "(" << p.first << " : " << p.second << ")";

return stream;

}

istream& operator >>(istream& stream, Pair& p)

{

cout << "Введите число first и second через пробел: ";

stream >> p.first >> p.second;

return stream;

}

bool Pair::operator >(const Pair& p)

{

if (this->first >= p.first && this->second > p.second)

{

return true;

}

if (this->first > p.first && this->second >= p.second)

{

return true;

}

return false;

}

bool Pair::operator <(const Pair& p)

{

if (this->first <= p.first && this->second < p.second)

{

return true;

}

if (this->first < p.first && this->second <= p.second)

{

return true;

}

return false;

}

bool operator==(const Pair& pair1, const Pair& pair2)

{

return pair1.first == pair2.first && pair1.second == pair2.second;

}

void Pair::operator/(const int& x)

{

this->first = this->first / x;

this->second = this->second / x;

}

bool operator<(const Pair& pair1, const Pair& pair2)

{

if (pair1.first <= pair2.first && pair1.first < pair2.first)

{

return true;

}

if (pair1.first < pair2.first && pair1.second <= pair2.second)

{

return true;

}

return false;

}

bool operator>(const Pair& pair1, const Pair& pair2)

{

if (pair1.first >= pair2.first && pair1.first > pair2.first)

{

return true;

}

if (pair1.first > pair2.first && pair1.second >= pair2.second)

{

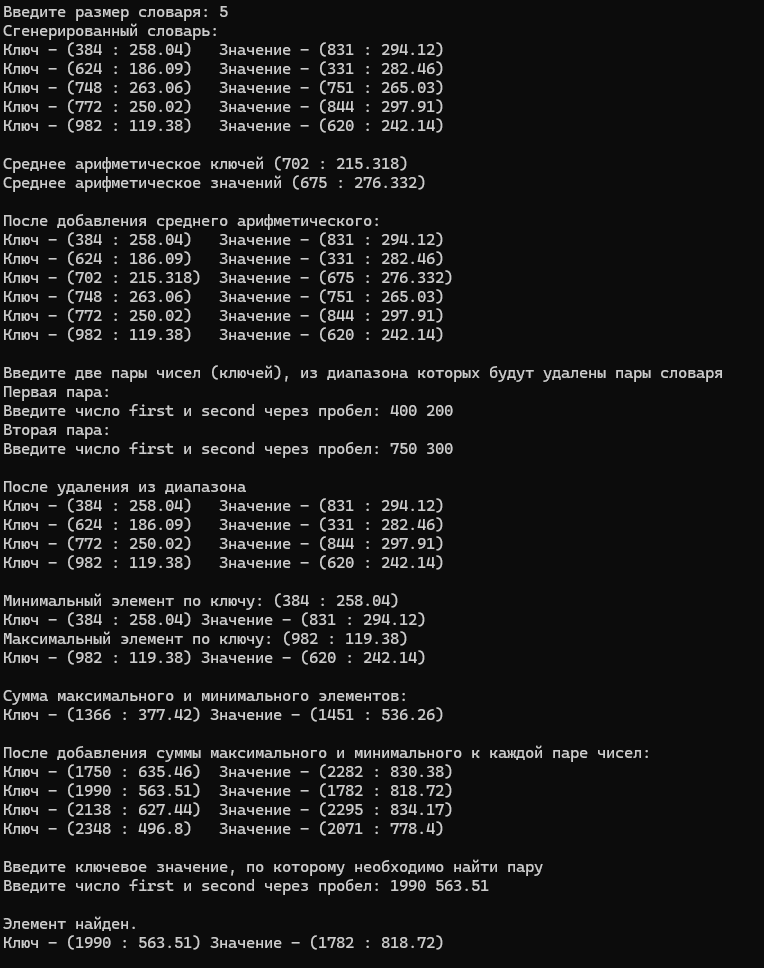
return true;

}

return false;

}

**Результаты работы**



**GitHub**

<https://github.com/Korovay4ik/Laboratory-works>