Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа по ООП №13**

**«Стандартные обобщенные алгоритмы библиотеки STL»**

Выполнил:

студент группы РИС-23-1б

Кривошеин Александр Антонович

Проверила:

доцент кафедры ИТАС

Полякова Ольга Андреевна

2024 г.

**Разработка алгоритма**

**Постановка задачи:**

**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

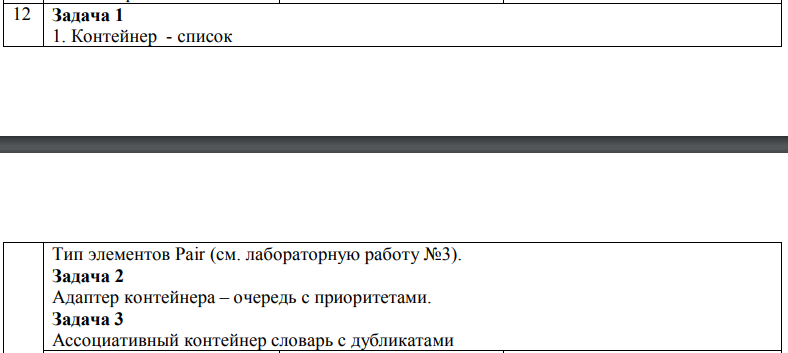
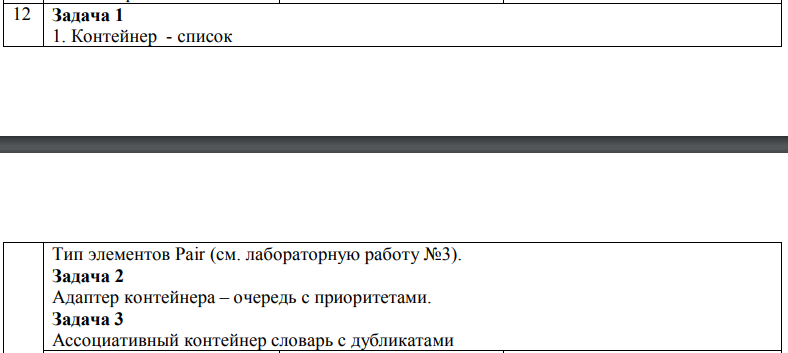
remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

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

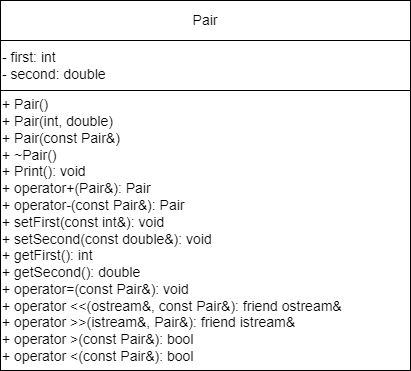
(использовать алгоритмы find(), find\_if(), count(), count\_if()).

1. Выполнить задание варианта для полученного контейнера

(использовать алгоритм for\_each()) .

****

**UML диаграмма**



**Код на C++**

main.cpp:

#include <list>

#include "pair.h"

#include "List.h"

#include <functional>

#include <iostream>

using namespace std;

int main() {

    double sum1 = 0;

    double sum2 = 0;

    List my\_list;

    int temp\_int;

    Pair temp\_pair;

    Pair temp\_pair2;

    cout << "\nEnter amount of elements: ";

    cin >> temp\_int;

    my\_list.fill\_list(temp\_int);

    my\_list.print();

    // вычисление среднего арифметического

    std::for\_each(my\_list.begin(), my\_list.end(), [&sum1, &sum2](const Pair& p) {

        sum1 += p.getFirst();

        sum2 += p.getSecond();

    });

    double average1 = sum1 / temp\_int;

    double average2 = sum2 / temp\_int;

    temp\_pair.set(average1, average2);

    cout << "\nArithmetic mean of pairs: " << temp\_pair << endl;

    cout << "\nEnter the position to insert the arithmetic mean of pairs: ";

    cin >> temp\_int;

    my\_list.insert\_pair(temp\_int, temp\_pair);

    my\_list.print();

    cout << "\nEnter two pairs of numbers to remove from the list: ";

    cin >> temp\_pair >> temp\_pair2;

    if (temp\_pair2 < temp\_pair) {

        swap(temp\_pair, temp\_pair2);

    }

    auto s = remove\_if(my\_list.begin(), my\_list.end(), [&temp\_pair, &temp\_pair2](const Pair& p) {

        return (temp\_pair <= p) && (p <= temp\_pair2);

    });

    my\_list.erase(s, my\_list.end());

    my\_list.print();

    cout << "\nAdding arithmetic mean to each element:\n";

    std::for\_each(my\_list.begin(), my\_list.end(), [&average1, &average2](Pair& p) {

        p.set(p.getFirst() + average1, p.getSecond() + average2);

    });

    my\_list.print();

    cout << "\nSort in Descending order:";

    my\_list.sort([](const Pair& a, const Pair& b) { return a > b; });

    my\_list.print();

    cout << "\nSort in Ascending order:";

    my\_list.sort([](const Pair& a, const Pair& b) { return a < b; });

    my\_list.print();

    cout << "\nEnter the value to find the pair: ";

    cin >> temp\_pair;

    temp\_int = 0;

    s = find\_if(my\_list.begin(), my\_list.end(), [&temp\_pair, &temp\_int](const Pair& p) {

        temp\_int++;

        return temp\_pair == p;

    });

    if (s != my\_list.end()) {

        cout << "Element has been found, its number: " << temp\_int << endl;

    } else {

        cout << "Element not found." << endl;

    }

}

List.h:

#pragma once

#include <iostream>

#include <list>

#include "pair.h"

using namespace std;

class List : public list<Pair> {

public:

    void print() const {

        if (empty()) {

            cout << "List is empty!\n";

            return;

        }

        for (const auto& pair : \*this) {

            cout << pair << ' ';

        }

        cout << endl;

    }

    void fill\_list(int n) {

        Pair temp\_pair;

        for (int i = 0; i < n; i++) {

            cin >> temp\_pair;

            push\_back(temp\_pair);

        }

    }

    void insert\_pair(int k, const Pair& temp\_pair) {

        auto iter = begin();

        advance(iter, k - 1);

        insert(iter, temp\_pair);

    }

};

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() = default;

    int getFirst() const;

    double getSecond() const;

    bool operator<(const Pair&) const;

    bool operator>(const Pair&) const;

    bool operator<=(const Pair&) const;

    bool operator>=(const Pair&) const;

    Pair& operator++();

    Pair& operator++(int);

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

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

    Pair& operator=(const Pair&);

    bool operator==(const Pair&) const;

    void set(int, double);

};

inline int Pair::getFirst() const {

    return first;

}

inline double Pair::getSecond() const {

    return second;

}

inline Pair::Pair() : first(0), second(0) {}

inline Pair::Pair(int fir, double sec) : first(fir), second(sec) {}

inline Pair::Pair(const Pair& copying) : first(copying.first), second(copying.second) {}

inline bool Pair::operator<(const Pair& temp) const {

    return (first + second) < (temp.first + temp.second);

}

inline bool Pair::operator<=(const Pair& temp) const {

    return (first + second) <= (temp.first + temp.second);

}

inline bool Pair::operator>(const Pair& temp) const {

    return (first + second) > (temp.first + temp.second);

}

inline bool Pair::operator>=(const Pair& temp) const {

    return (first + second) >= (temp.first + temp.second);

}

inline Pair& Pair::operator++() {

    ++first;

    return \*this;

}

inline Pair& Pair::operator++(int) {

    ++second;

    return \*this;

}

inline void Pair::set(int fir, double sec) {

    first = fir;

    second = sec;

}

inline ostream& operator<<(ostream& pairout, const Pair& temp) {

    return (pairout << temp.first << ':' << temp.second);

}

inline istream& operator>>(istream& stream, Pair& pair) {

    stream >> pair.first >> pair.second;

    return stream;

}

inline Pair& Pair::operator=(const Pair& other) {

    if (this != &other) {

        first = other.first;

        second = other.second;

    }

    return \*this;

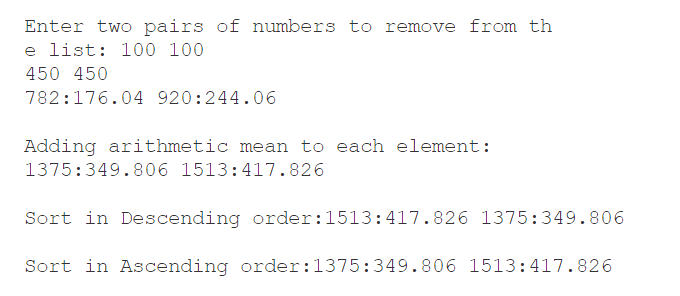
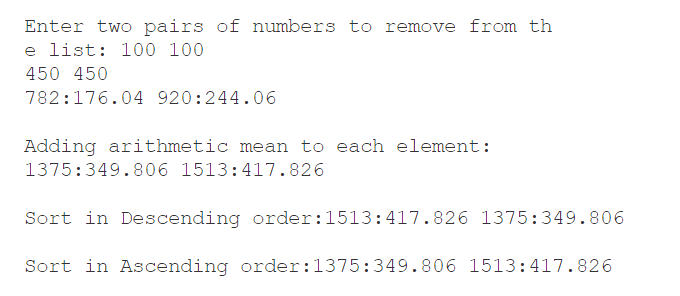
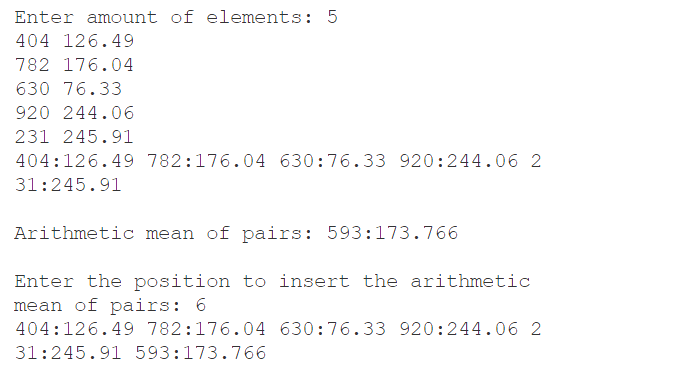
}

inline bool Pair::operator==(const Pair& other) const {

    return (first == other.first && second == other.second);

}

**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

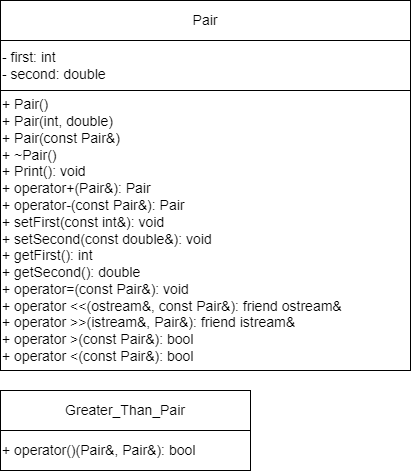
различные операции с объектами класса Pair. Например, оператор + позволяет складывать целое число с первым полем объекта класса и вещественное число со вторым полем соответственно. Оператор << позволяет выводить объекты класса Pair в поток вывода.

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 на C++**

**main.cpp:**

#include <iostream>

#include <queue>

#include <functional>

#include <algorithm>

#include "Pair.h"

#include "P\_queue.h"

#include "time.h"

using namespace std;

ostream &operator<<(ostream &, priority\_queue<Pair, vector<Pair>, Pair\_increase>);

Pair all\_sum(priority\_queue<Pair, vector<Pair>, Pair\_increase>);

int main()

{

    srand(time(0));

    int q\_size;

    cout << "Enter size of queue: ";

    cin >> q\_size;

    P\_queue my\_queue;

    my\_queue.generate\_Q(q\_size);

    cout << my\_queue;

    Pair sum = all\_sum(my\_queue);

    Pair average(sum.getFirst() / my\_queue.size(), sum.getSecond() / my\_queue.size());

    cout << "\nArithmetic mean: " << average << "\n\n";

    my\_queue.push(sum);

    cout << "After adding the arithmetic mean:\n"

         << my\_queue << "\n";

    Pair first\_p, second\_p;

    cout << "Enter two pairs of numbers from which the list items will be removed:\n";

    cout << "First pair:\n";

    cin >> first\_p;

    cout << "Second pair:\n";

    cin >> second\_p;

    if (first\_p > second\_p)

    {

        swap(first\_p, second\_p);

    }

    my\_queue.remove\_from\_range(first\_p, second\_p);

    cout << "\nAfter being removed from the range:\n"

         << my\_queue << "\n";

    my\_queue.substr\_to\_all(average);

    cout << "After subtracting the arithmetic mean " << average << " :\n"

         << my\_queue;

    cout << "Enter the pair value by which you want to find the pairу: ";

    Pair tmp\_pair;

    cin >> tmp\_pair;

    int index = my\_queue.find\_pair(tmp\_pair);

    if (index)

        cout << "The item has been found, its number: " << index + 1 << "\n";

    else

        cout << "Element not found. \n";

    return 0;

}

Pair all\_sum(priority\_queue<Pair, vector<Pair>, Pair\_increase> 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;

}

**P\_queue:**

#pragma once

#include <queue>

#include "pair.h"

#include <functional>

class Pair\_increase

{

public:

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

    {

        return p1 < p2;

    }

};

class P\_queue : public std::priority\_queue<Pair, vector<Pair>, Pair\_increase>

{

private:

    void randomize(Pair &p)

    {

        int x = rand() % 1001;

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

        p = Pair(x, y);

    }

public:

    void generate\_Q(int size)

    {

        Pair tmp(0, 0);

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

        {

            randomize(tmp);

            this->push(tmp);

        }

    }

    void remove\_from\_range(Pair &range\_1, Pair &range\_2)

    {

        P\_queue queue\_1;

        while (!this->empty())

        {

            Pair top\_pair = this->top();

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

            {

                queue\_1.push(top\_pair);

            }

            this->pop();

        }

        \*this = queue\_1;

    }

    void substr\_to\_all(Pair &pair)

    {

        P\_queue queue\_1;

        Pair top\_pair = this->top();

        while (!this->empty())

        {

            top\_pair = this->top();

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

            this->pop();

        }

        \*this = queue\_1;

    }

    int find\_pair(Pair &need\_p)

    {

        Pair tmp\_pair;

        int cnt = this->size();

        int index = -1;

        while (!this->empty())

        {

            tmp\_pair = this->top();

            if (tmp\_pair == need\_p)

            {

                index = cnt;

                return index;

            }

            this->pop();

            cnt--;

        }

        return index;

    }

    friend ostream &operator<<(ostream &stream, priority\_queue<Pair, vector<Pair>, Pair\_increase> queue)

    {

        while (!queue.empty())

        {

            Pair user = queue.top();

            cout << user << endl;

            queue.pop();

        }

        return stream;

    }

};

**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(){};

    int getFirst();

    double getSecond();

    bool operator<(Pair &);

    bool operator>(Pair &);

    bool operator<=(Pair &);

    bool operator>=(Pair &);

    Pair &operator++();

    Pair &operator+(const Pair &);

    Pair &operator-(const Pair &);

    Pair &operator++(int);

    friend ostream &operator<<(ostream &out, Pair i);

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

    void operator=(const Pair &);

    bool operator==(const Pair &);

    void get();

    void set(int, double);

};

int Pair::getFirst()

{

    return this->first;

}

double Pair::getSecond()

{

    return this->second;

}

Pair::Pair()

{

    this->first = 0;

    this->second = 0;

}

Pair::Pair(int fir, double sec)

{

    this->first = fir;

    this->second = sec;

}

Pair::Pair(const Pair &copying)

{

    this->first = copying.first;

    this->second = copying.second;

}

bool Pair::operator<(Pair &temp)

{

    if ((this->first + this->second) < (temp.first + temp.second))

        return true;

    return false;

}

bool Pair::operator<=(Pair &temp)

{

    if ((this->first + this->second) <= (temp.first + temp.second))

        return true;

    return false;

}

bool Pair::operator>(Pair &temp)

{

    if ((this->first + this->second) > (temp.first + temp.second))

        return true;

    return false;

}

bool Pair::operator>=(Pair &temp)

{

    if ((this->first + this->second) >= (temp.first + temp.second))

        return true;

    return false;

}

Pair &Pair::operator++()

{

    ++this->first;

    return \*this;

}

Pair &Pair::operator++(int)

{

    ++this->second;

    return \*this;

}

void Pair::get()

{

    std::cout << "first = " << this->first << ", second = " << this->second;

}

void Pair::set(int fir, double sec)

{

    this->first = fir;

    this->second = sec;

}

ostream &operator<<(ostream &pairout, Pair temp)

{

    return (pairout << temp.first << ':' << temp.second);

}

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

{

    int first;

    double second;

    stream >> first >> second;

    pair.first = first;

    pair.second = second;

    return stream;

}

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

{

    this->first = other.first;

    this->second = other.second;

}

bool Pair::operator==(const Pair &other)

{

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

        return true;

    else

        return false;

}

Pair &Pair::operator+(const Pair & add\_pair)

{

    this->first += add\_pair.first;

    this->second += add\_pair.second;

    return \*this;

}

Pair &Pair::operator-(const Pair & subst\_pair)

{

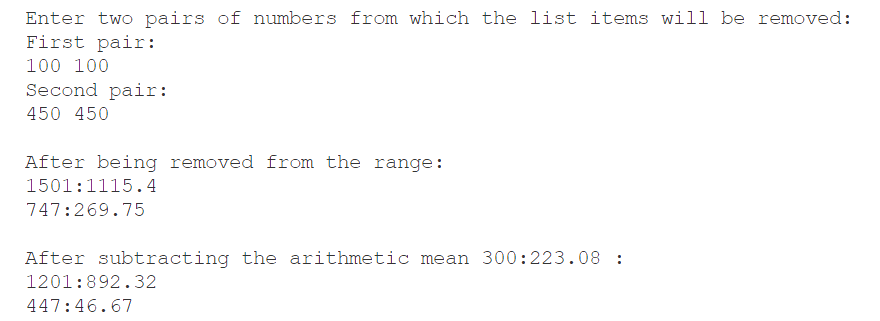
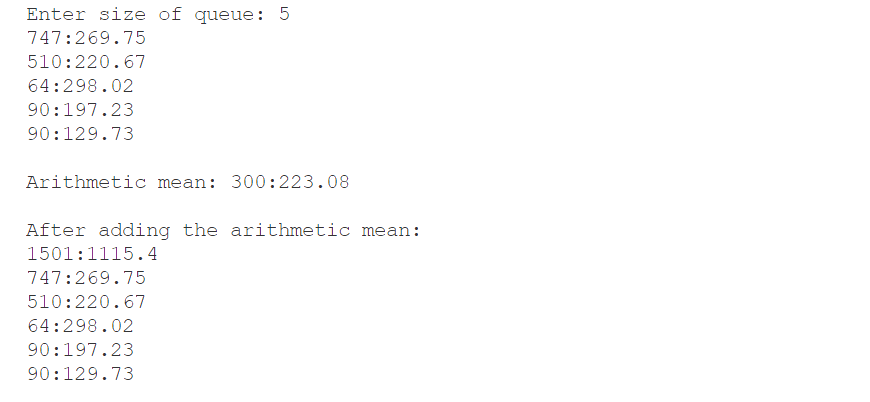
    this->first -= subst\_pair.first;

    this->second -= subst\_pair.second;

    return \*this;

}

**Результаты работы**



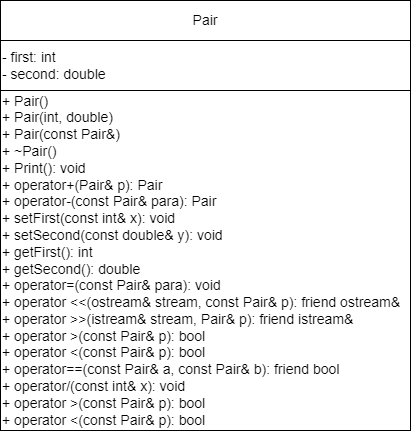
**Анализ задачи 3**

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 на C++**

**main.cpp:**

#include <iostream>

#include <functional>

#include "pair.h"

#include <map>

#include <time.h>

class Map : public map<Pair, Pair>

{};

using namespace std;

void randomize(Pair &);

void generate\_M(Map &, int);

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

ostream &operator<<(ostream &, const Map &);

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

void remove\_from\_range(Map &, Pair &, Pair &);

void all\_sum(Map &, Pair &, Pair &);

Map substr\_to\_all(Map &, Pair &, Pair &);

bool findPair(Map &, Pair &, Pair &);

int main()

{

    srand(time(0));

    int map\_size;

    cout << "Enter the size of the dictionary: ";

    cin >> map\_size;

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

    Map my\_map;

    generate\_M(my\_map, map\_size);

    cout << "Generated dictionary:\n"

         << my\_map << "\n";

    all\_sum(my\_map, sum\_1, sum\_2);

    Pair average\_key(sum\_1.getFirst() / my\_map.size(), sum\_1.getSecond() / my\_map.size());

    Pair average\_value(sum\_2.getFirst() / my\_map.size(), sum\_2.getSecond() / my\_map.size());

    cout << "The arithmetic mean of the keys " << average\_key << "\n";

    cout << "The arithmetic mean of the values " << average\_value << "\n\n";

    my\_map[average\_key] = average\_value;

    cout << "Adding an arithmetic mean:\n"

         << my\_map << "\n";

    Pair temp\_first\_p, temp\_second\_p;

    cout << "Enter two pairs of numbers (keys), from the range of which the dictionary elements will be removed:\n";

    cin >> temp\_first\_p >> temp\_second\_p;

    if (temp\_first\_p > temp\_second\_p)

    {

        swap(temp\_first\_p, temp\_second\_p);

    }

    remove\_from\_range(my\_map, temp\_first\_p, temp\_second\_p);

    cout << "\nResult:\n"

         << my\_map << "\n";

    cout << "The arithmetic mean of the keys " << average\_key << "\n";

    cout << "The arithmetic mean of the values " << average\_value << "\n\n";

    my\_map = substr\_to\_all(my\_map, average\_key, average\_value);

    cout << "After subtracting the arithmetic mean from each dictionary element:\n";

    cout << my\_map << "\n";

    cout << "Enter the key value by which you want to find the pair:\n";

    cin >> temp\_first\_p;

    bool found = findPair(my\_map, temp\_first\_p, temp\_second\_p);

    cout << "\n";

    if (found)

    {

        cout << "The element is found.\n";

        cout << temp\_first\_p << " -> " << temp\_second\_p << "\n";

    }

    else

    {

        cout << "The element was not found.\n";

    }

    return 0;

}

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

{

    stream  << p.first << "\t-> " << p.second << "\n";

    return stream;

}

ostream &operator<<(ostream &stream, const Map &t\_map)

{

    Map::const\_iterator Iter = t\_map.begin();

    while (Iter != t\_map.end())

    {

        cout << \*Iter++;

    }

    return stream;

}

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

{

    return key > lower && key < upper;

}

void remove\_from\_range(Map &t\_map, Pair &range\_1, Pair &range\_2)

{

    auto iterator = t\_map.begin();

    Pair key;

    while (iterator != t\_map.end())

    {

        key = iterator->first;

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

        {

            ++iterator;

        }

        else

        {

            iterator = t\_map.erase(iterator);

        }

    }

}

void all\_sum(Map &t\_map, Pair &key, Pair &value)

{

    for (const auto &t\_pair : t\_map)

    {

        Pair tmp = t\_pair.first;

        key = key + tmp;

        tmp = t\_pair.second;

        value = value + tmp;

    }

}

Map substr\_to\_all(Map &t\_map, Pair &key, Pair &value)

{

    Map tmp\_map;

    for (const auto &t\_pair : t\_map)

    {

        Pair tmp\_key = t\_pair.first;

        Pair tmp\_value = t\_pair.second;

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

    }

    return tmp\_map;

}

bool findPair(Map &t\_map, Pair &key, Pair &value)

{

    auto iter = t\_map.find(key);

    if (iter != t\_map.end())

    {

        value = Pair(iter->second);

        return true;

    }

    else

    {

        return false;

    }

}

void randomize(Pair &p)

{

    int x = rand() % 1001;

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

    p = Pair(x, y);

}

void generate\_M(Map &t\_map, 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);

        t\_map[qwe\_1] = qwe\_2;

    }

}

**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&);

};

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)

{

    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)

    {

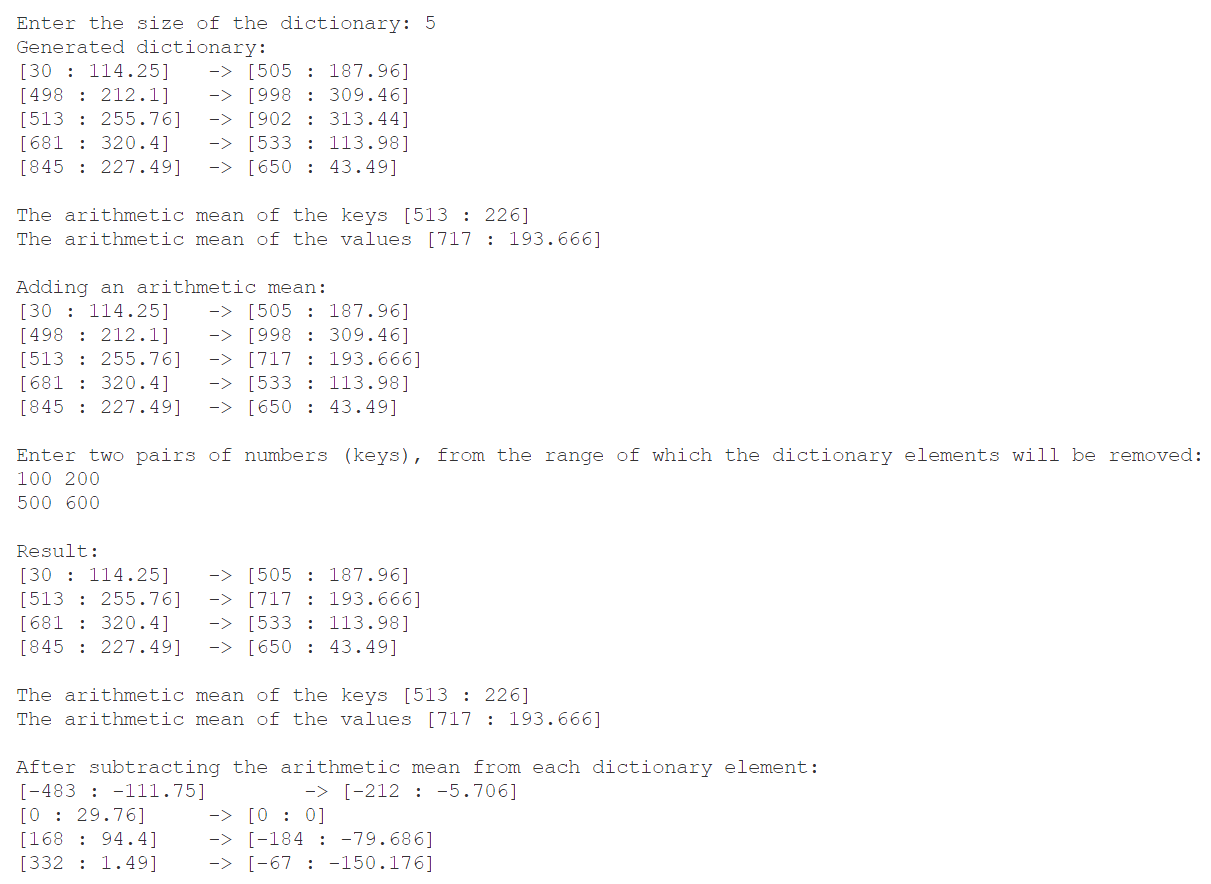
        return true;

    }

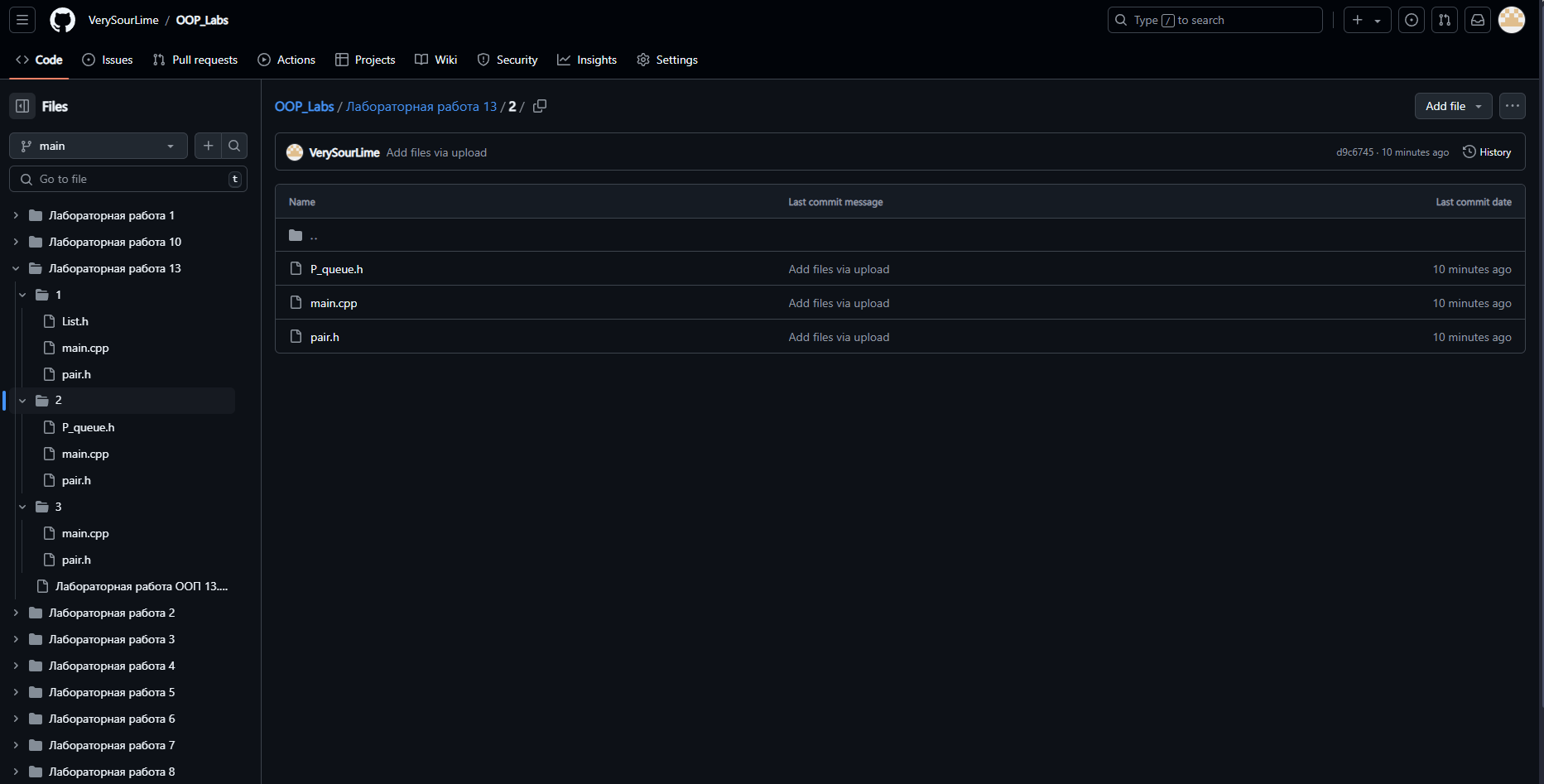
    return false;

}

**Результаты работы**



Скриншот из GitHub:

  
GitHub: <https://github.com/VerySourLime/OOP_Labs>