Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №11.1**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: Информационные и динамические структуры. Одно- и Двунаправленные списки Вариант 17

Выполнил:

Студент группы Рис-20-1б

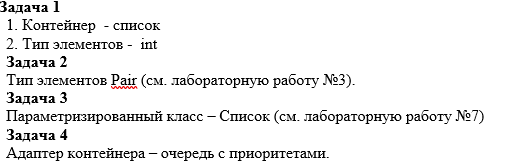
Томилов Владислав Алексеевич

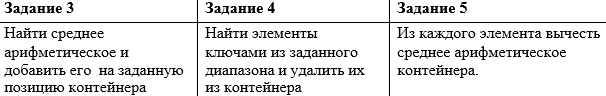
Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь, 2021**





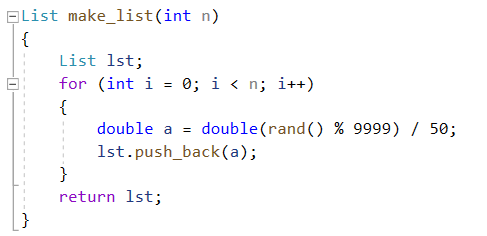
**11.1 Анализ задачи**

**1.** Для решения задачи необходимо:

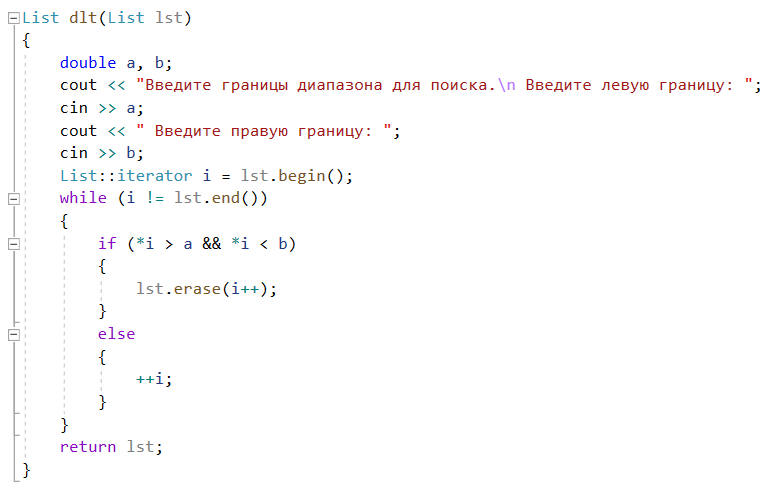
**1.1.** Создать контейнер List, в котором будет информационное поле

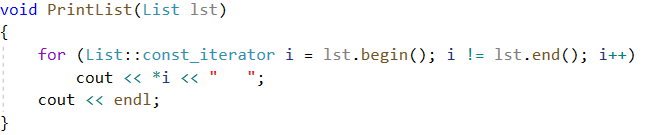
typedef list<double> List;

**1.2.** Разработать функцию создания списка makelist. Список заполняется рандомными значениями.

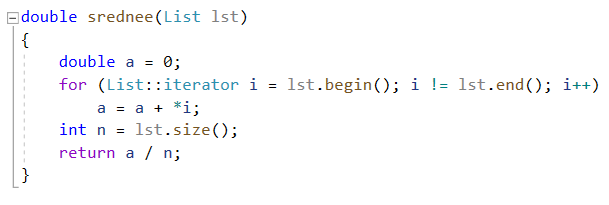


**1.3.**Разработать функцию dlt, которая будет проверять принадлежность элементов заданному промежутку значений и удалять те, которые вне этого промежутка.

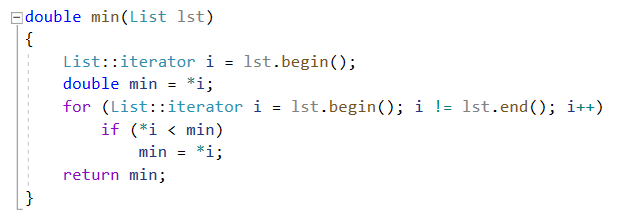


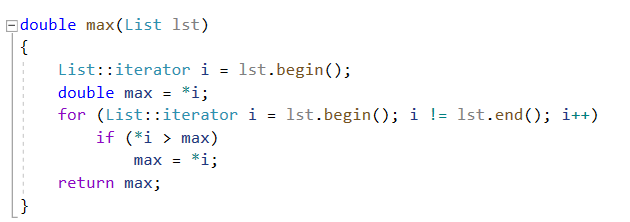
**1.4.** Разработать функцию printlist, которая будет выводить элементы списка в консоль.

**1.5.**Разработать функцию srednee, которая будет вычислять среднее арифметическое всех элементов списка

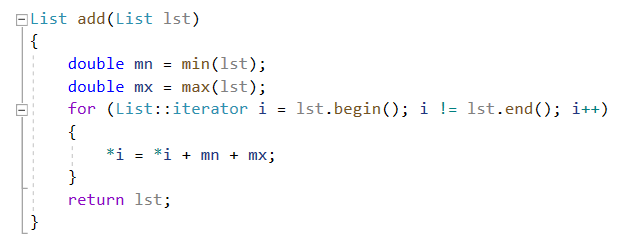


**1.6.** Разработать функцию min, которая будет находить минимальный элемент среди элементов списка.



**1.7.** Разработать функцию max, которая будет находить максимальный элемент среди элементов списка.

1.8. Разработать функцию add, которая добавляет min и max элементы в конец списка



**2.**В ходе работы были использованы следующие типы данных:

**2.1.** Для функции make\_list используются следующие аргументы:

1. Тип int: Размер списка, который пользователь будет вводить в консоль.

List make\_list(int n)

**2.4.** Для функции printlist пользуются следующие аргументы:

1. Тип список

void PrintList(List lst)

Сама функция имеет тип void, так как в ходе ее работы нет необходимости возвращать значение для этой функции.

**2.5.** Для функции srednee используются следующие аргументы:

1. Тип список

double srednee(List lst)

**2.6.** Для функции min используются следующие аргументы:

1. Тип список

double min(List lst)

**2.7.** Для функции max используются следующие аргументы:

1. Тип список

double max(List lst)

**2.8.** Для функции add используются следующие аргументы:

1. Тип список

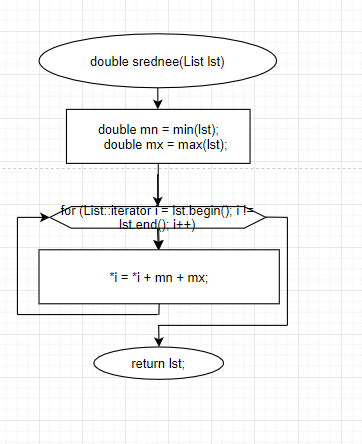
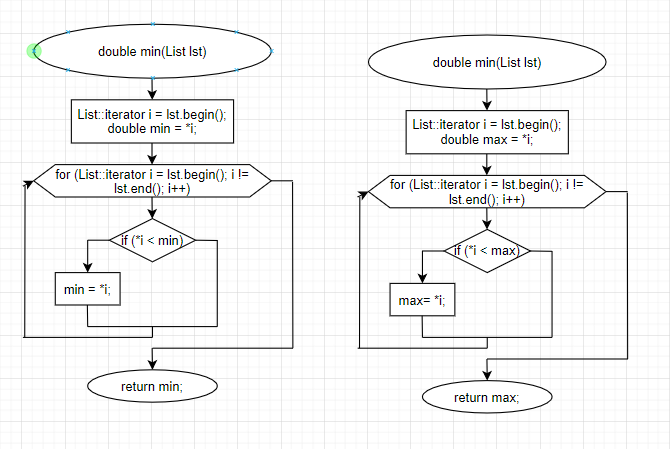
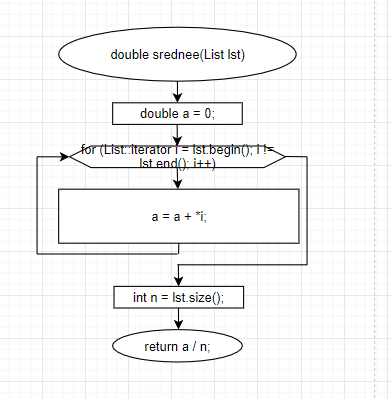
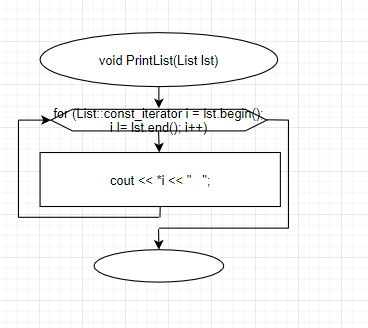
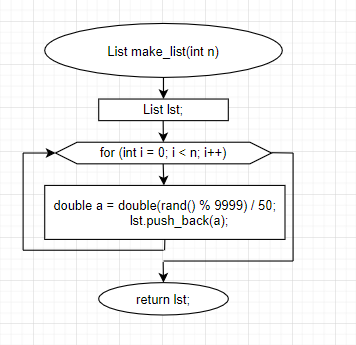
List add(List lst)

**2.9.** Для функции dlt используются следующие аргументы:

1. Тип список

List dlt(List lst)

**Блок-схема**



**Код программы**

#include <iostream>

#include <list>

#include <cstdlib>

using namespace std;

typedef list<double> List;

List make\_list(int n)

{

List lst;

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

{

double a = double(rand() % 9999) / 50;

lst.push\_back(a);

}

return lst;

}

void PrintList(List lst)

{

for (List::const\_iterator i = lst.begin(); i != lst.end(); i++)

cout << \*i << " ";

cout << endl;

}

double srednee(List lst)

{

double a = 0;

for (List::iterator i = lst.begin(); i != lst.end(); i++)

a = a + \*i;

int n = lst.size();

return a / n;

}

double min(List lst)

{

List::iterator i = lst.begin();

double min = \*i;

for (List::iterator i = lst.begin(); i != lst.end(); i++)

if (\*i < min)

min = \*i;

return min;

}

double max(List lst)

{

List::iterator i = lst.begin();

double max = \*i;

for (List::iterator i = lst.begin(); i != lst.end(); i++)

if (\*i > max)

max = \*i;

return max;

}

List add(List lst)

{

double mn = min(lst);

double mx = max(lst);

for (List::iterator i = lst.begin(); i != lst.end(); i++)

{

\*i = \*i + mn + mx;

}

return lst;

}

List dlt(List lst)

{

double a, b;

cout << "Введите границы диапазона для поиска.\n Введите левую границу: ";

cin >> a;

cout << " Введите правую границу: ";

cin >> b;

List::iterator i = lst.begin();

while (i != lst.end())

{

if (\*i > a && \*i < b)

{

lst.erase(i++);

}

else

{

++i;

}

}

return lst;

}

void main()

{

setlocale(LC\_ALL, "ru");

try

{

list<double> MyList;

list<double>::iterator it = MyList.begin();

int n;

cout << "Введите количество элементов в списке: ";

cin >> n;

MyList = make\_list(n);

PrintList(MyList);

cout << "\t3 задание\n";

cout << "Среднее арифметическое элементов списка = " << srednee(MyList) << endl;

MyList.push\_back(srednee(MyList));

PrintList(MyList);

cout << "\t4 задание\n";

MyList = dlt(MyList);

PrintList(MyList);

cout << "\t5 задание\n";

cout << "Минимальный элемент = " << min(MyList);

cout << "\tМаксимальный элемент = " << max(MyList);

MyList = add(MyList);

cout << "\nИзмененный список: ";

PrintList(MyList);

}

catch (int)

{

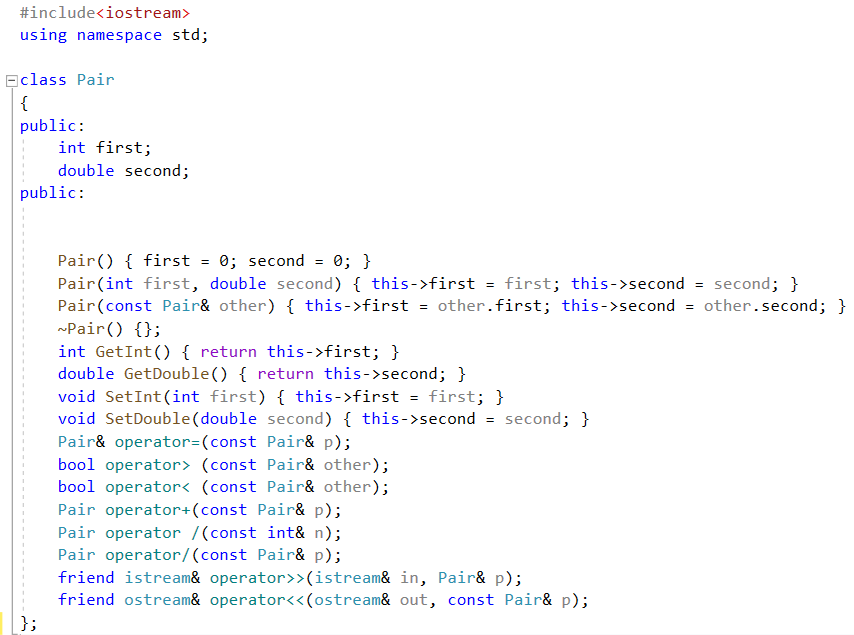
cout << "Error!";

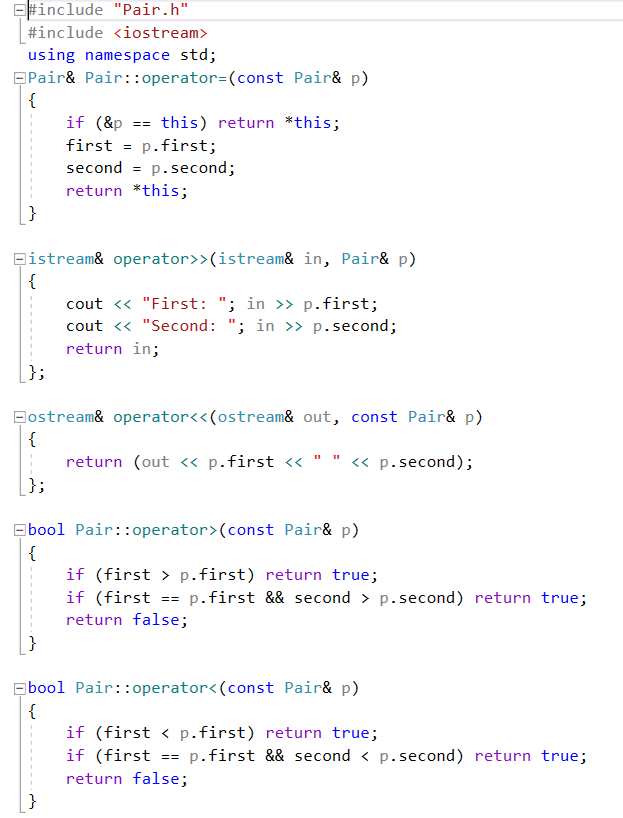
}

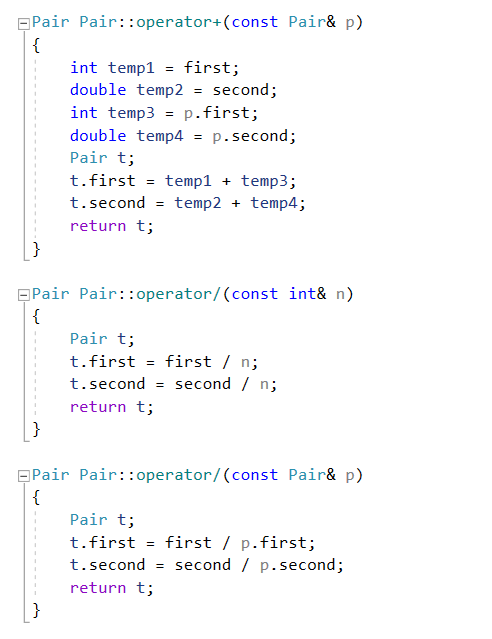
}

**11.2**

* 1. Для решения задачи необходимо
  2. Добавить в проект файл Pair.h, содержащий описание класса.



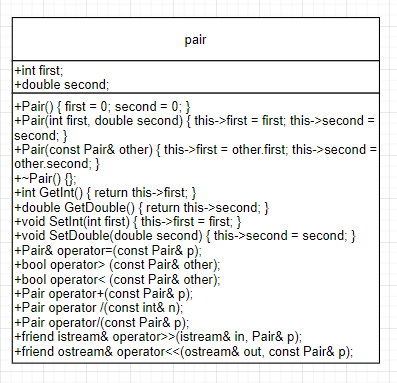
* 1. Добавить в проект файл Pair.cpp, содержащий описание методов класса.



* 1. Добавить в проект файл Lab11\_2\_main.cpp, содержащий основную программу.

Требуется заимствовать все ранее описанные функции в lab11.1

UML – диаграмма



Блок-схемы функций также заимствуются из lab11.1

**Код программы**

**Main.cpp**

#include <iostream>

#include <list>

#include <cstdlib>

#include "Pair.h"

using namespace std;

typedef list<Pair>List;

List make\_list(int n)

{

List lst;

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

{

Pair a;

a.first = rand() % 999;

a.second = double(rand() % 999) / 10;

lst.push\_back(a);

}

return lst;

}

void PrintList(List lst)

{

for (List::const\_iterator i = lst.begin(); i != lst.end(); i++)

cout << \*i << " ";

cout << endl;

}

List srednee(List lst)

{

int sfrst = 0;

double sscnd = 0;

for (List::iterator i = lst.begin(); i != lst.end(); i++)

{

sscnd = sscnd + (\*i).second;

sfrst = sfrst + (\*i).first;

}

int n = lst.size();

Pair a;

a.first = sfrst / n;

a.second = sscnd / n;

cout << "Среднее арифметическое значение элементов first= " << a.first << " , second= " << a.second << ".\n";

lst.push\_back(a);

return lst;

}

Pair min(List lst)

{

List::iterator i = lst.begin();

Pair min = \*i;

for (List::iterator i = lst.begin(); i != lst.end(); i++)

if (\*i < min)

min = \*i;

return min;

}

Pair max(List lst)

{

List::iterator i = lst.begin();

Pair max = \*i;

for (List::iterator i = lst.begin(); i != lst.end(); i++)

if (\*i > max)

max = \*i;

return max;

}

List add(List lst)

{

for (List::iterator i = lst.begin(); i != lst.end(); i++)

{

\*i = \*i + min(lst) + max(lst);

}

return lst;

}

List dlt(List lst)

{

double a, b;

cout << "Введите границы диапазона для поиска.\n Введите левую границу: ";

cin >> a;

cout << " Введите правую границу: ";

cin >> b;

List::iterator i = lst.begin();

int k = lst.size();

while (i != lst.end())

{

if (((\*i).first > a && (\*i).first < b) || (\*i).second > a && (\*i).second < b)

{

lst.erase(i++);

k--;

}

else

{

++i;

}

}

if (k == 0) { cout << "Пустой список"; List lst; exit; }

else return lst;

}

void main()

{

setlocale(LC\_ALL, "RU");

try

{

list<Pair> MyList;

list<Pair>::iterator it = MyList.begin();

int n;

cout << "Введите количество элементов в списке: ";

cin >> n;

MyList = make\_list(n);

PrintList(MyList);

MyList = srednee(MyList);

cout << "Список после добавления элемента со САЗ в конец:\n";

PrintList(MyList);

MyList = dlt(MyList);

cout << "Список после удаления элементов из заданного диапазона:\n";

PrintList(MyList);

cout << "Минимальный элемент списка= " << min(MyList) << " , максимальный = " << max(MyList) << endl;

MyList = add(MyList);

cout << "Список после добавления значения минимального и максимального элемента к каждому в списке:\n";

PrintList(MyList);

}

catch (int)

{

cout << "Error!";

}

}

**Pair.h**

#pragma once

#include<iostream>

using namespace std;

class Pair

{

public:

int first;

double second;

public:

Pair() { first = 0; second = 0; }

Pair(int first, double second) { this->first = first; this->second = second; }

Pair(const Pair& other) { this->first = other.first; this->second = other.second; }

~Pair() {};

int GetInt() { return this->first; }

double GetDouble() { return this->second; }

void SetInt(int first) { this->first = first; }

void SetDouble(double second) { this->second = second; }

Pair& operator=(const Pair& p);

bool operator> (const Pair& other);

bool operator< (const Pair& other);

Pair operator+(const Pair& p);

Pair operator /(const int& n);

Pair operator/(const Pair& p);

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

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

};

**Pair.cpp**

#include "Pair.h"

#include <iostream>

using namespace std;

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

{

if (&p == this) return \*this;

first = p.first;

second = p.second;

return \*this;

}

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

{

cout << "First: "; in >> p.first;

cout << "Second: "; in >> p.second;

return in;

};

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

{

return (out << p.first << " " << p.second);

};

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

{

if (first > p.first) return true;

if (first == p.first && second > p.second) return true;

return false;

}

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

{

if (first < p.first) return true;

if (first == p.first && second < p.second) return true;

return false;

}

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

{

int temp1 = first;

double temp2 = second;

int temp3 = p.first;

double temp4 = p.second;

Pair t;

t.first = temp1 + temp3;

t.second = temp2 + temp4;

return t;

}

Pair Pair::operator/(const int& n)

{

Pair t;

t.first = first / n;

t.second = second / n;

return t;

}

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

{

Pair t;

t.first = first / p.first;

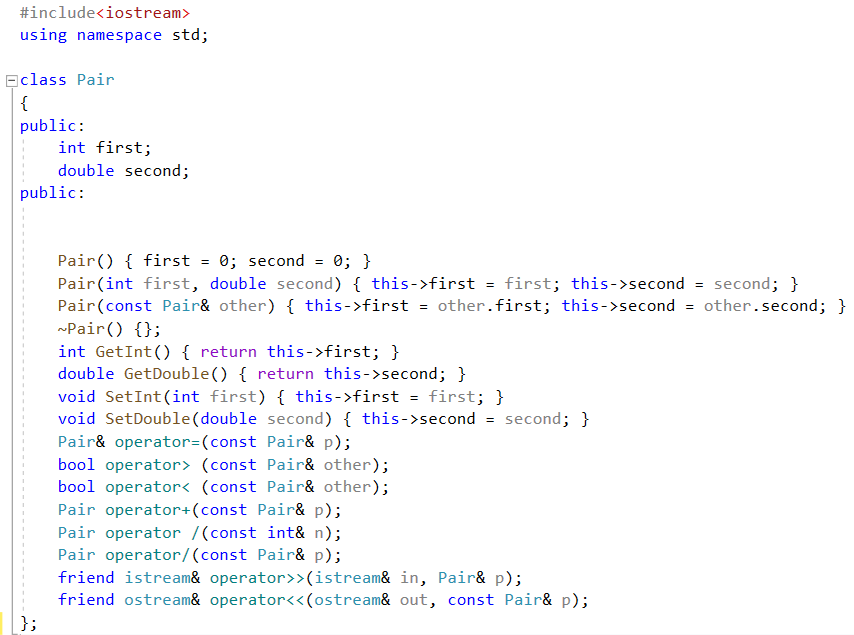
t.second = second / p.second;

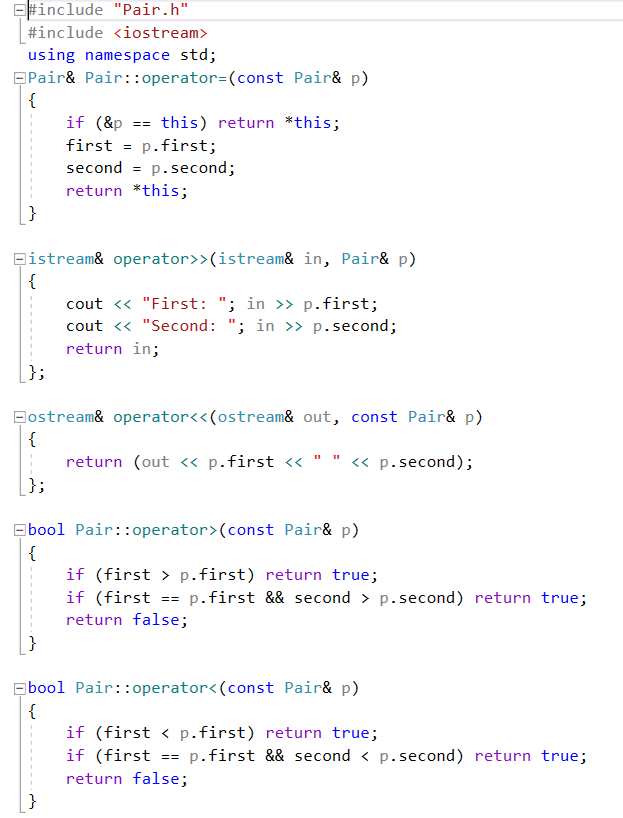
return t;

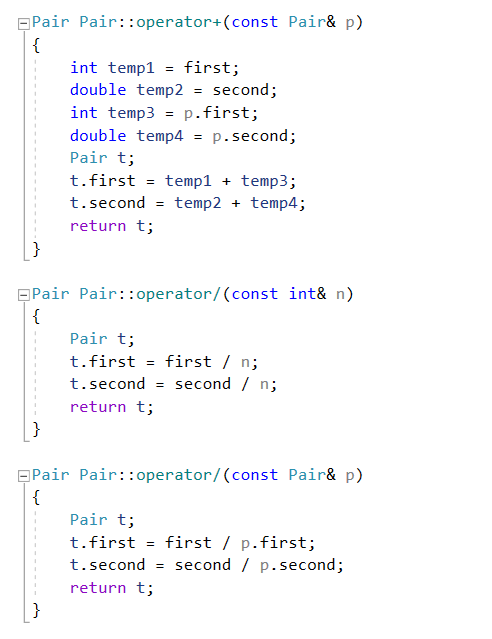
}

**11.3**

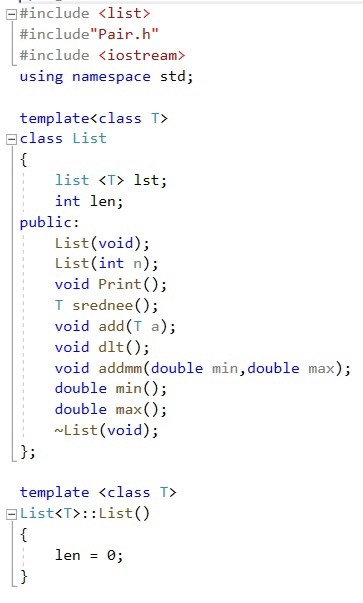
* 1. Для решения задачи необходимо
  2. Добавить в проект файл Pair.h, содержащий описание класса.

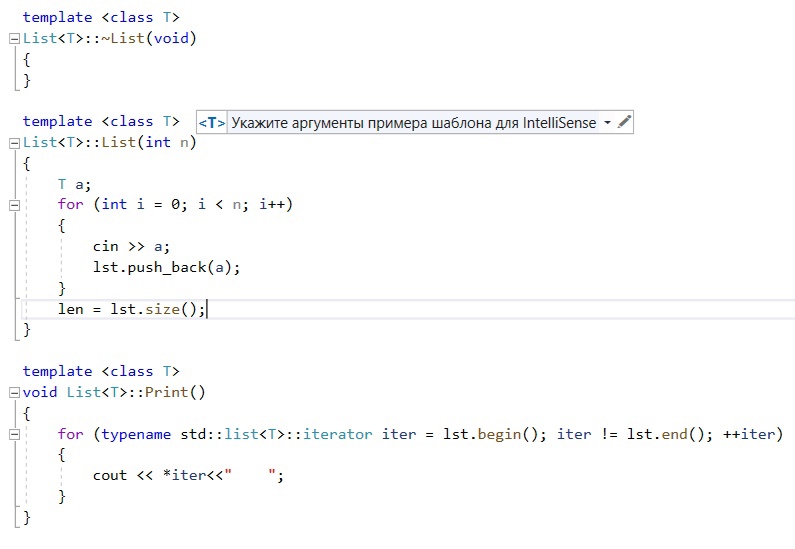
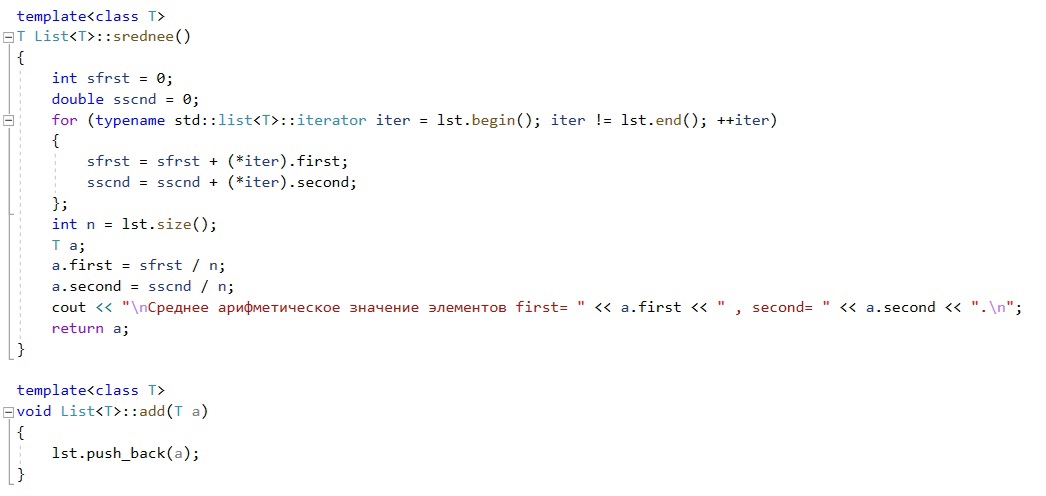


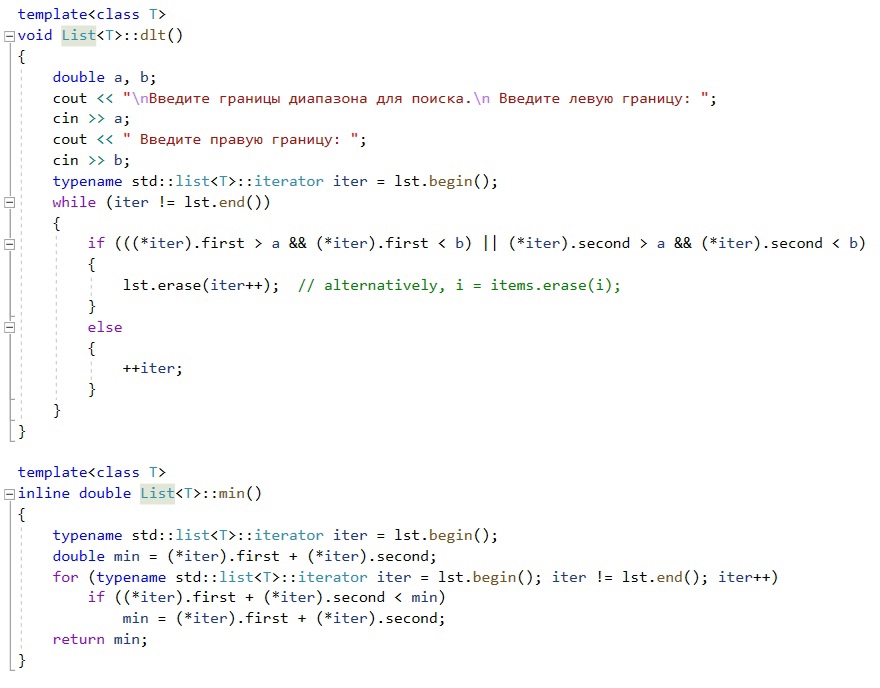
* 1. Добавить в проект файл Pair.cpp, содержащий описание методов класса.

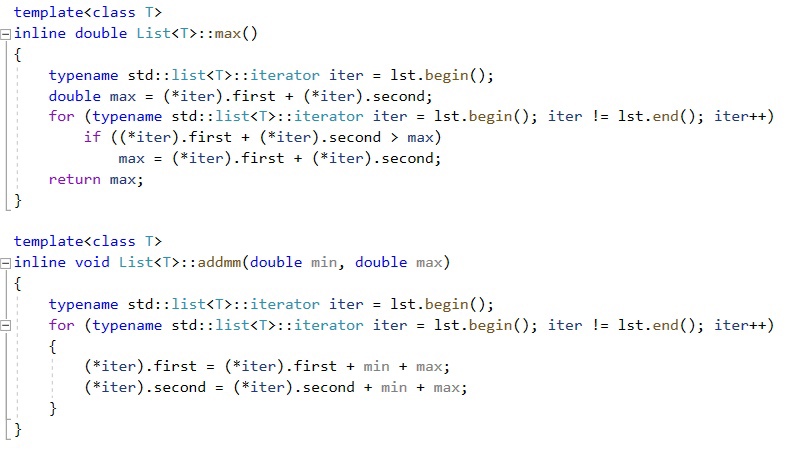


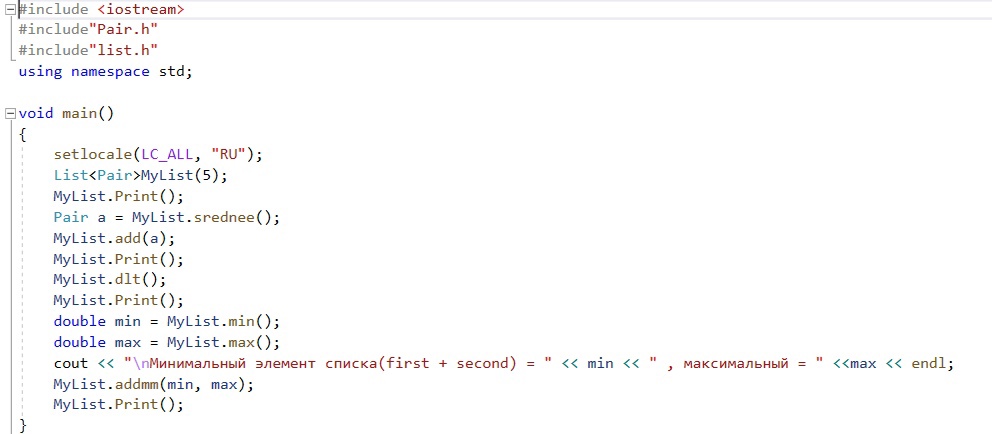
* 1. Добавить в проект файл list.h, содержащий описание класса и описание его методов



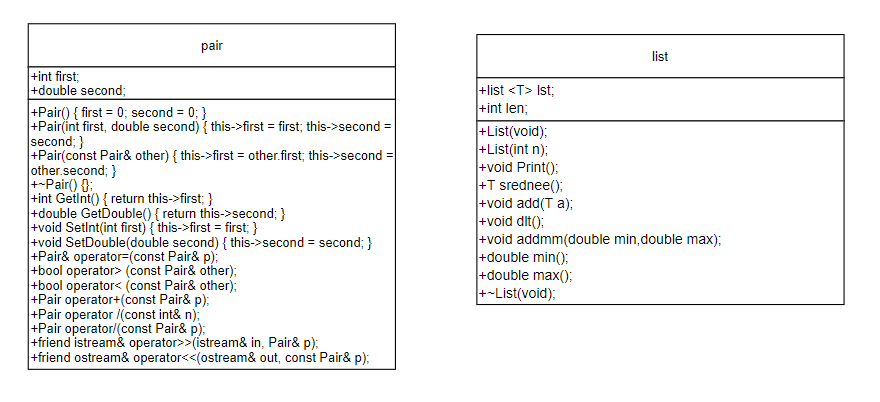






* 1. Добавить в проект файл Lab11\_3\_main.cpp, содержащий основную программу.

UML – диаграмма



Код программы

Main.cpp

* [#include](https://vk.com/im?sel=223733620&st=%23include) <iostream>  
  [#include](https://vk.com/im?sel=223733620&st=%23include)"Pair.h"  
  [#include](https://vk.com/im?sel=223733620&st=%23include)"list.h"  
  using namespace std;  
    
  void main()  
  {  
  setlocale(LC\_ALL, "RU");  
  List<Pair>MyList(5);  
  MyList.Print();  
  Pair a = MyList.srednee();  
  MyList.add(a);  
  MyList.Print();  
  MyList.dlt();  
  MyList.Print();  
  double min = MyList.min();  
  double max = MyList.max();  
  cout « "\nМинимальный элемент списка(first + second) = " « min « " , максимальный = " «max « endl;  
  MyList.addmm(min, max);  
  MyList.Print();  
  }

Pair.h

#pragma once

#include<iostream>

using namespace std;

class Pair

{

public:

int first;

double second;

public:

Pair() { first = 0; second = 0; }

Pair(int first, double second) { this->first = first; this->second = second; }

Pair(const Pair& other) { this->first = other.first; this->second = other.second; }

~Pair() {};

int GetInt() { return this->first; }

double GetDouble() { return this->second; }

void SetInt(int first) { this->first = first; }

void SetDouble(double second) { this->second = second; }

Pair& operator=(const Pair& p);

bool operator> (const Pair& other);

bool operator< (const Pair& other);

Pair operator+(const Pair& p);

Pair operator /(const int& n);

Pair operator/(const Pair& p);

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

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

};

Pair.cpp

#include "Pair.h"

#include <iostream>

using namespace std;

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

{

if (&p == this) return \*this;

first = p.first;

second = p.second;

return \*this;

}

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

{

cout << "First: "; in >> p.first;

cout << "Second: "; in >> p.second;

return in;

};

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

{

return (out << p.first << " " << p.second);

};

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

{

if (first > p.first) return true;

if (first == p.first && second > p.second) return true;

return false;

}

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

{

if (first < p.first) return true;

if (first == p.first && second < p.second) return true;

return false;

}

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

{

int temp1 = first;

double temp2 = second;

int temp3 = p.first;

double temp4 = p.second;

Pair t;

t.first = temp1 + temp3;

t.second = temp2 + temp4;

return t;

}

Pair Pair::operator/(const int& n)

{

Pair t;

t.first = first / n;

t.second = second / n;

return t;

}

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

{

Pair t;

t.first = first / p.first;

t.second = second / p.second;

return t;

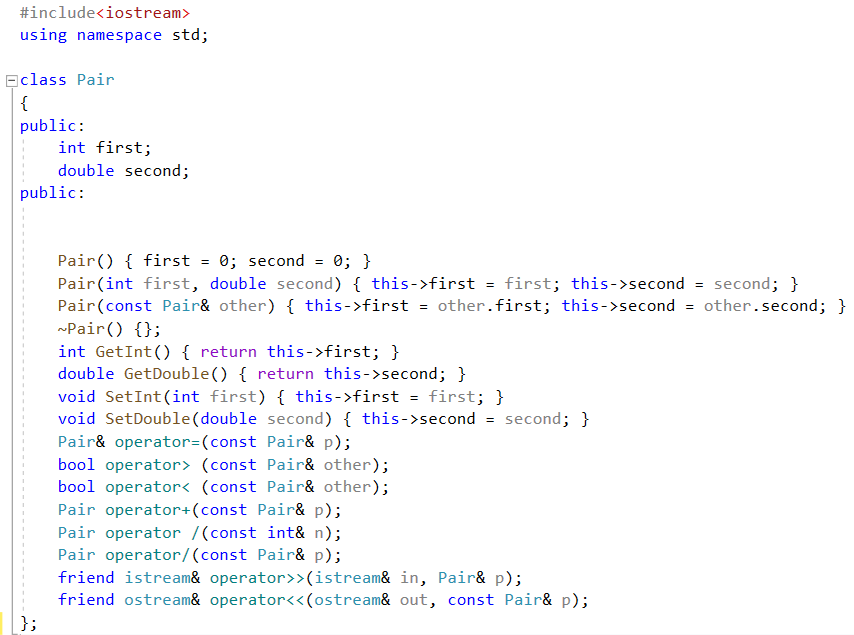
}

List.h

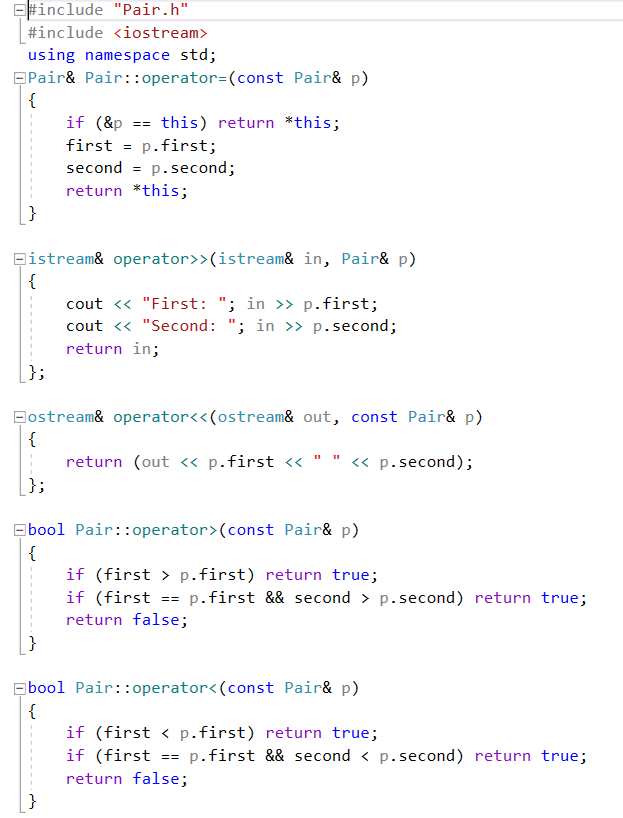
[#include](https://vk.com/im?sel=223733620&st=%23include) <list>  
[#include](https://vk.com/im?sel=223733620&st=%23include)"Pair.h"  
[#include](https://vk.com/im?sel=223733620&st=%23include) <iostream>  
using namespace std;  
  
template<class T>  
class List  
{  
list <T> lst;  
int len;  
public:  
List(void);  
List(int n);  
void Print();  
T srednee();  
void add(T a);  
void dlt();  
void addmm(double min,double max);  
double min();  
double max();  
~List(void);  
};  
  
template <class T>  
List<T>::List()  
{  
len = 0;  
}  
  
  
  
template <class T>  
List<T>::~List(void)  
{  
}  
  
template <class T>  
List<T>::List(int n)  
{  
T a;  
for (int i = 0; i < n; i++)  
{  
cin » a;  
lst.push\_back(a);  
}  
len = lst.size();  
}  
  
template <class T>  
void List<T>::Print()  
{  
for (typename std::list<T>::iterator iter = lst.begin(); iter != lst.end(); ++iter)  
{  
cout « \*iter«" ";  
}  
}  
  
template<class T>  
T List<T>::srednee()  
{  
int sfrst = 0;  
double sscnd = 0;  
for (typename std::list<T>::iterator iter = lst.begin(); iter != lst.end(); ++iter)  
{  
sfrst = sfrst + (\*iter).first;  
sscnd = sscnd + (\*iter).second;  
};  
int n = lst.size();  
T a;  
a.first = sfrst / n;  
a.second = sscnd / n;  
cout « "\nСреднее арифметическое значение элементов first= " « a.first « " , second= " « a.second « ".\n";  
return a;  
}  
  
template<class T>  
void List<T>::add(T a)  
{  
lst.push\_back(a);  
}  
  
template<class T>  
void List<T>::dlt()  
{  
double a, b;  
cout « "\nВведите границы диапазона для поиска.\n Введите левую границу: ";  
cin » a;  
cout « " Введите правую границу: ";  
cin » b;  
typename std::list<T>::iterator iter = lst.begin();  
while (iter != lst.end())  
{  
if (((\*iter).first > a && (\*iter).first < b) || (\*iter).second > a && (\*iter).second < b)  
{  
lst.erase(iter++); // alternatively, i = items.erase(i);  
}  
else  
{  
++iter;  
}  
}  
}  
  
template<class T>  
inline double List<T>::min()  
{  
typename std::list<T>::iterator iter = lst.begin();  
double min = (\*iter).first + (\*iter).second;  
for (typename std::list<T>::iterator iter = lst.begin(); iter != lst.end(); iter++)  
if ((\*iter).first + (\*iter).second < min)  
min = (\*iter).first + (\*iter).second;  
return min;  
}  
  
  
template<class T>  
inline double List<T>::max()  
{  
typename std::list<T>::iterator iter = lst.begin();  
double max = (\*iter).first + (\*iter).second;  
for (typename std::list<T>::iterator iter = lst.begin(); iter != lst.end(); iter++)  
if ((\*iter).first + (\*iter).second > max)  
max = (\*iter).first + (\*iter).second;  
return max;  
}  
  
template<class T>  
inline void List<T>::addmm(double min, double max)  
{  
typename std::list<T>::iterator iter = lst.begin();  
for (typename std::list<T>::iterator iter = lst.begin(); iter != lst.end(); iter++)  
{  
(\*iter).first = (\*iter).first + min + max;  
(\*iter).second = (\*iter).second + min + max;  
}  
}

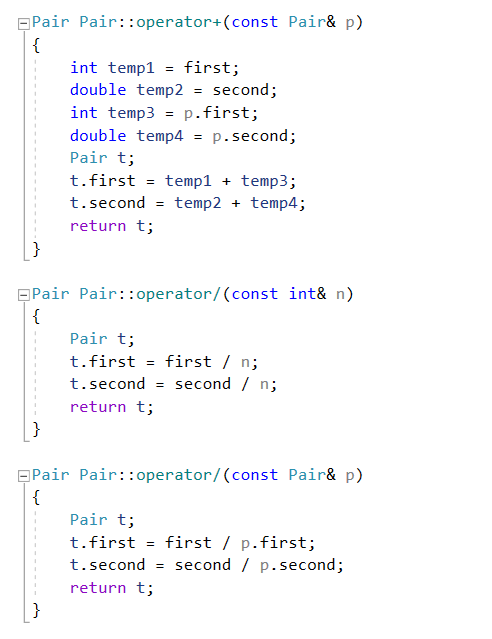
**11.4**

* 1. Для решения задачи необходимо
  2. Добавить в проект файл Pair.h, содержащий описание класса.



* 1. Добавить в проект файл Pair.cpp, содержащий описание методов класса.





1.3 Добавить в проект файл Lab11\_4\_main.cpp, содержащий основную программу

void main()

{

setlocale(LC\_ALL, "ru");

Pair p;

Qe s;

int n;

cout << "Введите количество элементов в списке: ";

cin >> n;

s = make\_queue(n);

print\_queue(s);

cout << "\nСреднее арифметическое значение элементов: " << average(s) << endl;

add(s);

print\_queue(s);

dlt(s);

print\_queue(s);

cout << "\nmin= " << min(s) << "\nmax= " << max(s) << endl;;

addmm(s);

print\_queue(s);

}

1.4 Переопределим все функции списка для очереди

1. Создание очереди

Qe make\_queue(int n)

{

Qe s;

Pair p;

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

{

cin >> p;

s.push(p);

}

return s;

}

1. Вывод очереди

void print\_queue(Qe s) {

while (!s.empty()) {

cout << s.front() << ' ';

s.pop();

}

cout << '\n';

}

1. Копирование из очереди в вспомогательный список

List copy\_queue\_to\_list(Qe s)

{

List a;

while (!s.empty())//пока очередь не пустая

{

a.push\_back(s.front());//добавление элемента из стека в список

s.pop();//убираем элемент с вершины стека

}

return a;

}

1. Копирование из вспомогательного списка в очередь

Qe copy\_list\_to\_queue(List a)

{

Qe s;

for (List::iterator iter = a.begin(); iter != a.end(); iter++)

{

s.push((\*iter));

}

return s;

1. Нахождение среднего арифметического

Qe copy\_list\_to\_queue(List a)

{

Qe s;

for (List::iterator iter = a.begin(); iter != a.end(); iter++)

{

s.push((\*iter));

}

return s;

}

1. Нахождение min/max

Pair max(Qe s)

{

List a = copy\_queue\_to\_list(s);

List::iterator iter = a.begin();

Pair max=(\*iter);

for (iter; iter != a.end(); iter++)

{

if (\*iter > max)

max = \*iter;

}

return max;

}

Pair min(Qe s)

{

List a = copy\_queue\_to\_list(s);

List::iterator iter = a.begin();

Pair min = (\*iter);

for (iter; iter != a.end(); iter++)

{

if (\*iter < min)

min= \*iter;

}

return min;

}

1. Добавление среднего арифметического в очередь

void add(Qe&s)

{

cout << "\nДобавление среднего арифметического в конец контейнера.\n";

s.push(average(s));

}

1. Удаление элементов непопадающих в заданный промежуток

void dlt(Qe &s)

{

cout << "\nУдаление элементов из заданного диапазона.\n";

List a = copy\_queue\_to\_list(s);

double x, b;

cout << "\nВведите границы диапазона для поиска.\n Введите левую границу: ";

cin >> x;

cout << " Введите правую границу: ";

cin >> b;

List::iterator i = a.begin();

while (i != a.end())

{

if (((\*i).first > x && (\*i).first < b) || (\*i).second > x && (\*i).second < b)

{

a.erase(i++); // alternatively, i = items.erase(i);

}

else

{

++i;

}

}

s = copy\_list\_to\_queue(a);

}

1. Добавление к каждому элементу суммы минимального и максимального элементов

void addmm(Qe &s)

{

cout << "\nДобавление к каждому элементу суммы минимального и максимального элементов контейнера.\n";

List a = copy\_queue\_to\_list(s);

for (List::iterator i = a.begin(); i != a.end(); i++)

{

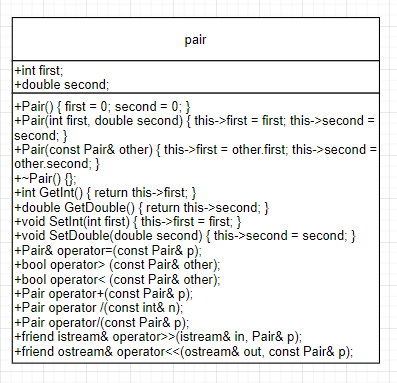
\*i =\*i+ min(s) + max(s);

}

s = copy\_list\_to\_queue(a);

}

UML – диаграмма



Код программы

**Main.cpp**

#include <functional>

#include <iostream>

#include"Pair.h"

#include <queue>

#include<list>

using namespace std;

typedef queue<Pair>Qe;

typedef list<Pair> List;

Qe make\_queue(int n)

{

Qe s;

Pair p;

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

{

cin >> p;

s.push(p);

}

return s;

}

void print\_queue(Qe s) {

while (!s.empty()) {

cout << s.front() << ' ';

s.pop();

}

cout << '\n';

}

List copy\_queue\_to\_list(Qe s)

{

List a;

while (!s.empty())//пока очередь не пустая

{

a.push\_back(s.front());//добавление элемента из стека в список

s.pop();//убираем элемент с вершины стека

}

return a;

}

Qe copy\_list\_to\_queue(List a)

{

Qe s;

for (List::iterator iter = a.begin(); iter != a.end(); iter++)

{

s.push((\*iter));

}

return s;

}

Pair average(Qe s)

{

List a = copy\_queue\_to\_list(s);

int n = 1;

Pair sum = s.front();

s.pop();

while (!s.empty())

{

sum = sum + s.front();

s.pop();

n++;

}

s = copy\_list\_to\_queue(a);

return sum / n;

}

void add(Qe&s)

{

cout << "\nДобавление среднего арифметического в конец контейнера.\n";

s.push(average(s));

}

void dlt(Qe &s)

{

cout << "\nУдаление элементов из заданного диапазона.\n";

List a = copy\_queue\_to\_list(s);

double x, b;

cout << "\nВведите границы диапазона для поиска.\n Введите левую границу: ";

cin >> x;

cout << " Введите правую границу: ";

cin >> b;

List::iterator i = a.begin();

while (i != a.end())

{

if (((\*i).first > x && (\*i).first < b) || (\*i).second > x && (\*i).second < b)

{

a.erase(i++); // alternatively, i = items.erase(i);

}

else

{

++i;

}

}

s = copy\_list\_to\_queue(a);

}

Pair max(Qe s)

{

List a = copy\_queue\_to\_list(s);

List::iterator iter = a.begin();

Pair max=(\*iter);

for (iter; iter != a.end(); iter++)

{

if (\*iter > max)

max = \*iter;

}

return max;

}

Pair min(Qe s)

{

List a = copy\_queue\_to\_list(s);

List::iterator iter = a.begin();

Pair min = (\*iter);

for (iter; iter != a.end(); iter++)

{

if (\*iter < min)

min= \*iter;

}

return min;

}

void addmm(Qe &s)

{

cout << "\nДобавление к каждому элементу суммы минимального и максимального элементов контейнера.\n";

List a = copy\_queue\_to\_list(s);

for (List::iterator i = a.begin(); i != a.end(); i++)

{

\*i =\*i+ min(s) + max(s);

}

s = copy\_list\_to\_queue(a);

}

void main()

{

setlocale(LC\_ALL, "ru");

Pair p;

Qe s;

int n;

cout << "Введите количество элементов в списке: ";

cin >> n;

s = make\_queue(n);

print\_queue(s);

cout << "\nСреднее арифметическое значение элементов: " << average(s) << endl;

add(s);

print\_queue(s);

dlt(s);

print\_queue(s);

cout << "\nmin= " << min(s) << "\nmax= " << max(s) << endl;;

addmm(s);

print\_queue(s);

}

**Pair.h**

#pragma once

#include<iostream>

using namespace std;

class Pair

{

public:

int first;

double second;

public:

Pair() { first = 0; second = 0; }

Pair(int first, double second) { this->first = first; this->second = second; }

Pair(const Pair& other) { this->first = other.first; this->second = other.second; }

~Pair() {};

int GetInt() { return this->first; }

double GetDouble() { return this->second; }

void SetInt(int first) { this->first = first; }

void SetDouble(double second) { this->second = second; }

Pair& operator=(const Pair& p);

bool operator> (const Pair& other);

bool operator< (const Pair& other);

Pair operator+(const Pair& p);

Pair operator /(const int& n);

Pair operator/(const Pair& p);

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

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

};

**Pair.cpp**

#include "Pair.h"

#include <iostream>

using namespace std;

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

{

if (&p == this) return \*this;

first = p.first;

second = p.second;

return \*this;

}

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

{

cout << "First: "; in >> p.first;

cout << "Second: "; in >> p.second;

return in;

};

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

{

return (out << p.first << " " << p.second);

};

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

{

if (first > p.first) return true;

if (first == p.first && second > p.second) return true;

return false;

}

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

{

if (first < p.first) return true;

if (first == p.first && second < p.second) return true;

return false;

}

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

{

int temp1 = first;

double temp2 = second;

int temp3 = p.first;

double temp4 = p.second;

Pair t;

t.first = temp1 + temp3;

t.second = temp2 + temp4;

return t;

}

Pair Pair::operator/(const int& n)

{

Pair t;

t.first = first / n;

t.second = second / n;

return t;

}

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

{

Pair t;

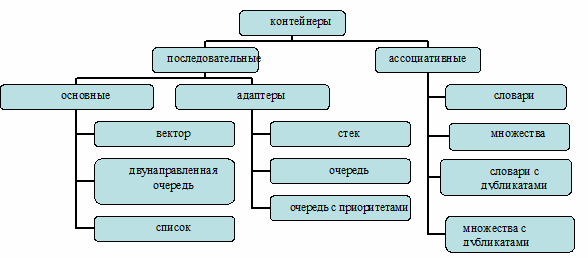
t.first = first / p.first;

t.second = second / p.second;

return t;

}

Ответы на вопросы

1. STL – Standard Template Library, стандартная библиотека шаблонов состоит из двух основных частей: набора контейнерных классов и набора обобщенных алгоритмов.
2. 
3. Для использования контейнера в программе необходимо включить в нее соот-

ветствующий заголовочный файл

1. Итератор более общее понятие, чем указатель. Тип iterator определен для всех

контейнерных классов STL, однако, реализация его в разных классах разная

* К основным операциям, выполняемым с любыми итераторами, относятся:
* Разыменование итератора: если р — итератор, то \*р — значение объекта, на который он ссылается.
* Присваивание одного итератора другому.
* Сравнение итераторов на равенство и неравенство (== и !=).
* Перемещение его по всем элементам контейнера с помощью префиксного (++р) или постфиксного (р++) инкремента.

Организация циклов просмотра элементов контейнеров тоже имеет некоторую специфику. Так, если i — некоторый итератор, то вместо привычной формы

for (i =0; i < n; ++i) используется следующая:

for (i = first; i != last; ++i),

где first - значение итератора, указывающее на первый элемент в контейнере, a last — значение итератора, указывающее на воображаемый элемент, который следует за последним элементом контейнера.

1. В STL существуют следующие типы итераторов:

* входные,
* выходные,
* прямые,
* двунаправленные итераторы,
* итераторы произвольного доступа.

|  |  |
| --- | --- |
| **Операция или метод** | **Пояснение** |
| Операции равенства (==) и  неравенства (!=) | Возвращают значение true или false |
| Операция присваивания (=) | Копирует один контейнер в другой |
| clear | Удаляет все элементы |
| insert | Добавляет один элемент или диапазон элементов |
| erase | Удаляет один элемент или диапазон элементов |
| size\_type size() const | Возвращает число элементов |
| size\_type max\_size() const | Возвращает максимально допустимый размер  контейнера |
| bool empty0 const | Возвращает true, если контейнер пуст |
| iterator begin() | Возвращают итератор на начало контейнера (итерации будут производиться в прямом  направлении) |
| iterator end() | Возвращают итератор на конец контейнера  (итерации в прямом направлении будут закончены) |
| reverse\_iterator begin() | Возвращают реверсивный итератор на конец  контейнера (итерации будут производиться в обратном направлении) |
| reverse\_iterator end() | Возвращают реверсивный итератор на начало  контейнера (итерации в обратном направлении будут закончены |

1. –
2. –
3. –

|  |  |
| --- | --- |
| Vector | |
| push\_back() | добавление  в конец |
| pop\_back() | удаление из  конца |
| insert | Вставка в  произвольн ое место |
| erase | удаление из произвольн  ог оместа |
| []  at | доступ к произвольн ому  элементу |
|  |  |
|  |  |
| swap | обмен |

|  |  |
| --- | --- |
| Deque | |
| push\_back(T&key) | добавление  в конец |
| pop\_back() | удаление из  конца |
| push\_front(T&key) | добавление в начало |
| pop\_front() | удаление из начала |
| insert | Вставка в произвольн ое место |
| erase | удаление из произвольн  ог оместа |
| []  at | доступ к произвольн ому  элементу |

|  |  |
| --- | --- |
| List | |
| push\_back(T&key) | добавление  в конец |
| pop\_back() | удаление из  конца |
| push\_front(T&key) | добавление в начало |
| pop\_front() | удаление из начала |
| insert | Вставка в произвольн ое место |
| erase | удаление из произвольн  ог оместа |
|  |  |
| swap | обмен |

1. –
2. pop\_back()
3. –
4. pop\_back()
5. –
6. pop\_back()
7. –
8. Специализированные последовательные контейнеры — стек, очередь и очередь с приоритетами — не являются самостоятельными контейнерными классами, а реализованы на основе рассмотренных выше классов, поэтому они называются адаптерами контейнеров.
9. Смысл такой реализации заключается в том, что специализированный класс просто переопределяет интерфейс класса-прототипа, ограничивая его только теми методами, которые нужны новому классу.
10. Методы класса stack:
    * + push () - добавление в конец;
      + pop () - удаление из конца;
      + top () - получение текущего элемента стека;
      + empty() - проверка пустой стек или нет;
      + size () – получение размера стека.
11. Методы класса queue:

* push () – добавление в конец очереди;
* pop () – удаление из начала очереди;
* front () – получение первого элемента очереди;
* back()- получение последнего элемента очереди;
* empty () – проверка пустая очередь или нет;
* size() – получение размера очереди.

1. Реализация класса возможна либо на основе вектора (реализация по умолчанию), либо на основе списка.
2. -
3. –
4. –
5. -