Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №11

Тема: «Последовательные контейнеры библиотеки STL.»

Выполнили:

Студент группы РИС-22-2б

Худеньких В.Д.

Проверил доц. Кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

# Постановка задачи

Задача 1.

* + 1. Создать последовательный контейнер.
    2. Заполнить его элементами стандартного типа (тип указан в варианте).
    3. Добавить элементы в соответствии с заданием
    4. Удалить элементы в соответствии с заданием.
    5. Выполнить задание варианта для полученного контейнера.
    6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 2.

1. Создать последовательный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 3

1. Создать параметризированный класс, используя в качестве контейнера последовательный контейнер.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

Задача 4

1. Создать адаптер контейнера.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 5

1. Создать параметризированный класс, используя в качестве контейнера адаптер контейнера.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

ВАРИАНТ 15:

Задача 1

1. Контейнер - список

2. Тип элементов - double

Задача 2

Тип элементов Pair (см. лабораторную работу №3).

Задача 3

Параметризированный класс – Список (см. лабораторную работу №7)

Задача 4

Адаптер контейнера – очередь с приоритетами.

Задача 5

Параметризированный класс – Список

Адаптер контейнера – очередь с приоритетами.

Задание 3

Найти среднее арифметическое и добавить его в конец контейнера

Задание 4

Найти элементы ключами из заданного диапазона и удалить их из контейнера

Задание 5

К каждому элементу добавить сумму минимального и максимального элементов контейнера.

**Контрольные вопросы**

1. Из каких частей состоит библиотека STL?

STL – *Standard Template Library*, стандартная библиотека шаблонов состоит из двух основных частей: набора контейнерных классов и набора обобщенных алгоритмов.

1. Какие типы контейнеров существуют в STL?

Контейнеры STL можно разделить на два типа: последовательные и ассоциативные

1. Что нужно сделать для использования контейнера STL в своей программе?

Для использования контейнера в программе необходимо включить в нее соответствующий заголовочный файл. Тип объектов, сохраняемых в контейнере, задается с помощью аргумента шаблона, например:

#include <vector>

#include <list>

#include “person.h”

…..

vector<int> v;

list<person> l;

1. Что представляет собой итератор?

Функцию можно использовать для поиска элементов в массиве любого типа, но использовать ее для списка нельзя, поэтому авторы STL ввели понятие итератора. Итератор более общее понятие, чем указатель. Тип iterator определен для всех контейнерных классов STL

1. Какие операции можно выполнять над итераторами?

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.

1. Каким образом можно организовать цикл для перебора контейнера с использованием итератора?

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.

Операция сравнения < заменена на операцию ! =, т. к. операции < и > для итераторов в общем случае не поддерживаются.

1. Какие типы итераторов существуют?

В STL существуют следующие типы итераторов:

• входные,

• выходные,

• прямые,

• двунаправленные итераторы,

• итераторы произвольного доступа.

1. Какие операции являются эффективными для контейнера vector? Почему?

Контейнер вектор является аналогом обычного массива, за исключением того, что он автоматически выделяет и освобождает память по мере необходимости. Контейнер эффективно обрабатывает произвольную выборку элементов с помощью операции индексации [] или метода at. Однако вставка элемента в любую позицию, кроме конца вектора, неэффективна. Для этого потребуется сдвинуть все последующие элементы путем копирования их значений. По этой же причине неэффективным является удаление любого элемента, кроме последнего.

1. Какие операции являются эффективными для контейнера list? Почему?

Контейнер список организует хранение объектов в виде двусвязного списка. Каждый элемент списка содержит три поля: значение элемента, указатель на предшествующий и указатель на последующий элементы списка. Вставка и удаление работают эффективно для любой позиции элемента в списке. Однако список не поддерживает произвольного доступа к своим элементам: например, для выборки n-го элемента нужно последовательно выбрать предыдущие п-1 элементов.

1. Какие операции являются эффективными для контейнера deque? Почему?

Контейнер двусторонняя очередь во многом аналогичен вектору, элементы хранятся в непрерывной области памяти. Но в отличие от вектора двусторонняя очередь эффективно поддерживает вставку и удаление первого элемента (так же, как и последнего).

1. Перечислить методы, которые поддерживает последовательный контейнер deque

|  |  |
| --- | --- |
| 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 |

1. Задан контейнер vector. Как удалить из него элементы со 2 по 5?

vector<int> l;

vector<int>::iterator it\_start = l.begin();

vector<int>::iterator it\_finish = l.begin();

advance(it\_start, 2); //смещение итератора до начальной точки отсчета

advance(it\_finish, 5); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

1. Задан контейнер vector. Как удалить из него последний элемент?

vector<double> l;

l.pop\_back();

1. Задан контейнер list. Как удалить из него элементы со 2 по 5?

list<string> l;

list<string>::iterator it\_start = l.begin();

list<string>::iterator it\_finish = l.begin();

advance(it\_start, 2); //смещение итератора до начальной точки отсчета

advance(it\_finish, 5); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

1. Задан контейнер list. Как удалить из него последний элемент?

list<double> l;

l.pop\_back();

1. Задан контейнер deque. Как удалить из него элементы со 2 по 5?

deque<int> l;

deque<int>::iterator it\_start = l.begin();

deque<int>::iterator it\_finish = l.begin();

advance(it\_start, 2); //смещение итератора до начальной точки отсчета

advance(it\_finish, 5); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

1. Задан контейнер deque. Как удалить из него последний элемент?

deque<double> l;

l.pop\_back();

1. Написать функцию для печати последовательного контейнера с использованием итератора.

void print\_list(LinkedList l)

{

cout << "\nСейчас список такой: \n";

copy(l.begin(), l.end(), ostream\_iterator<double>(cout, " "));

}

1. Что представляют собой адаптеры контейнеров?

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.

1. Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?

Объявление stack<int> s создает стек на базе двусторонней очереди (по умолчанию). Если по каким-то причинам нас это не устраивает, и мы хотим создать стек на базе списка, то объявление будет выглядеть следующим образом:

stack<int, list<int> > s;

Смысл такой реализации заключается в том, что специализированный класс просто переопределяет интерфейс класса-прототипа, ограничивая его только теми методами, которые нужны новому классу. Cтек не позволяет выполнить произвольный доступ к своим элементам, а также не дает возможности пошагового перемещения, т. е. итераторы в стеке не поддерживаются

1. Перечислить методы, которые поддерживает контейнер stack.

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.

1. Перечислить методы, которые поддерживает контейнер queue.

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. Чем отличаются друг от друга контейнеры queue и priority\_queue?

Шаблонный класс priority\_queue (заголовочный файл <queue>) поддерживает такие же операции, как и класс queue, но реализация класса возможна либо на основе вектора (реализация по умолчанию), либо на основе списка. Очередь с приоритетами отличается от обычной очереди тем, что для извлечения выбирается максимальный элемент из хранимых в контейнере. Поэтому после каждого изменения состояния очереди максимальный элемент из оставшихся сдвигается в начало контейнера.

1. Задан контейнер stack. Как удалить из него элемент с заданным номером?

template <class T>

void del(stack<T>& st, int index)

{

stack<T> temp;

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

{

temp.push(st.top());

st.pop();

}ss

st.pop();

while (temp.size())

{

st.push(temp.top());

temp.pop();

}

}

Но так делать не надо, потому что stack не предназначен для того, чтобы из него удалялись элементы не по порядку. В данном случае будет гораздо логичнее применить структуру данных список или любой контейнер STL.

1. Задан контейнер queue. Как удалить из него элемент с заданным номером?

std::deque<int> dq{1, 2, 3, 4, 5};

int index = 2;

auto it = std::next(dq.begin(), index);

dq.erase(it);

1. Написать функцию для печати контейнера stack с использованием итератора.

Это невозможно, так как у контейнера stack даже итераторов нет.

Вот пример как можно посмотреть все, что лежит в stack, но так делать тоже не надо, потому что это ломает весь смысл работы с данным контейнером, гораздо логичнее будет взять любой другой контейнер, например отлично подойдет однонаправленный список.

template <class T>

void printStack(stack<T> s)

{

while (!s.empty()) {

cout << s.top() << " ";

s.pop();

}}

1. Написать функцию для печати контейнера queue с использованием итератора.

template <class T>

void printDeque(deque<T> dq)

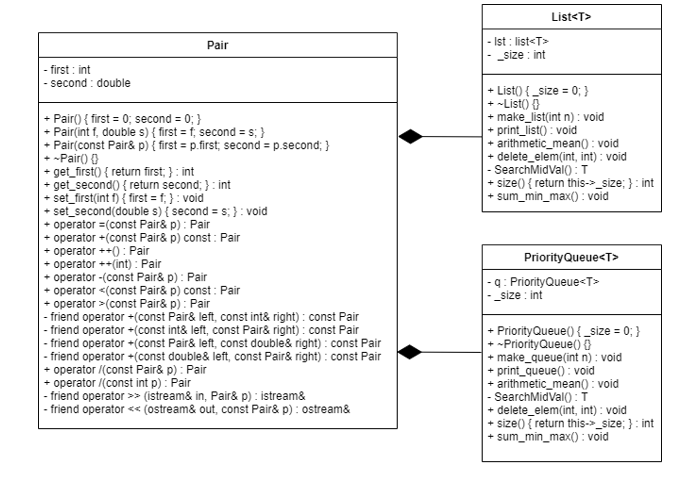
{

for (auto i = dq.begin(); i != dq.end(); ++i)

cout << \*i << endl;

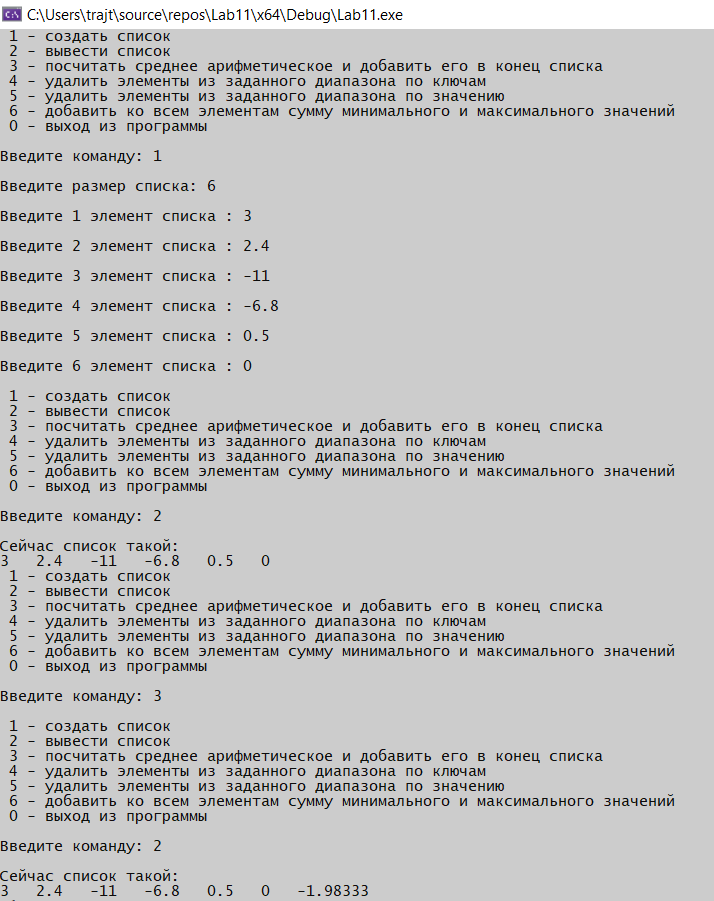
}

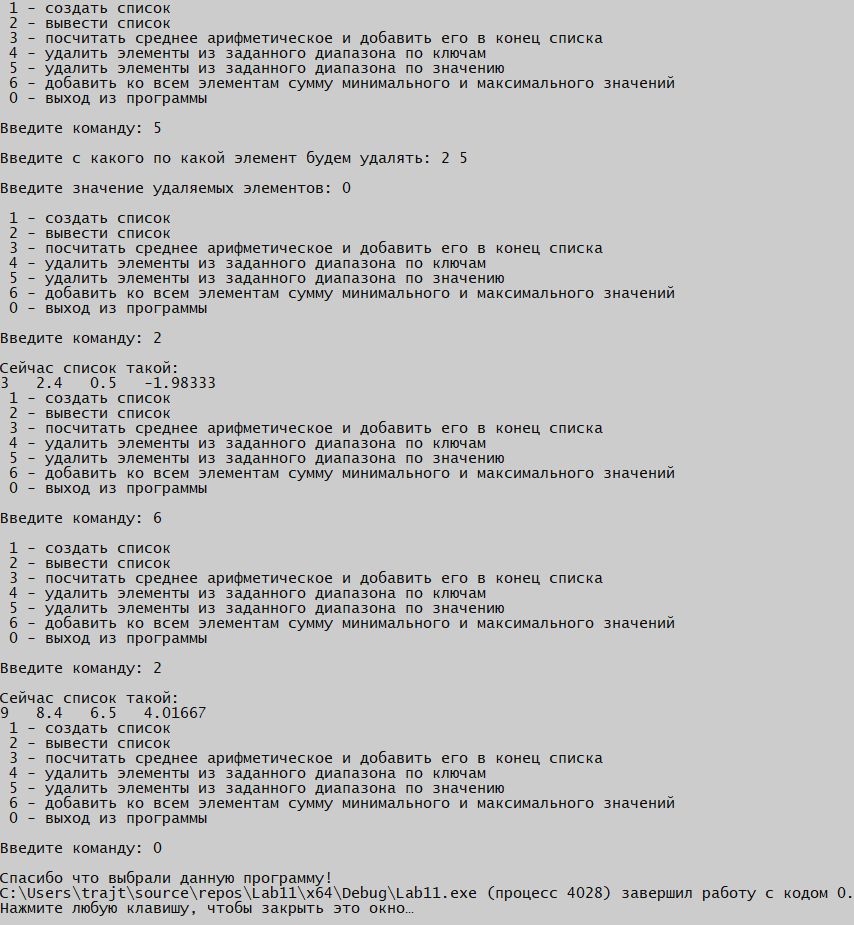
**UML таблица**

****

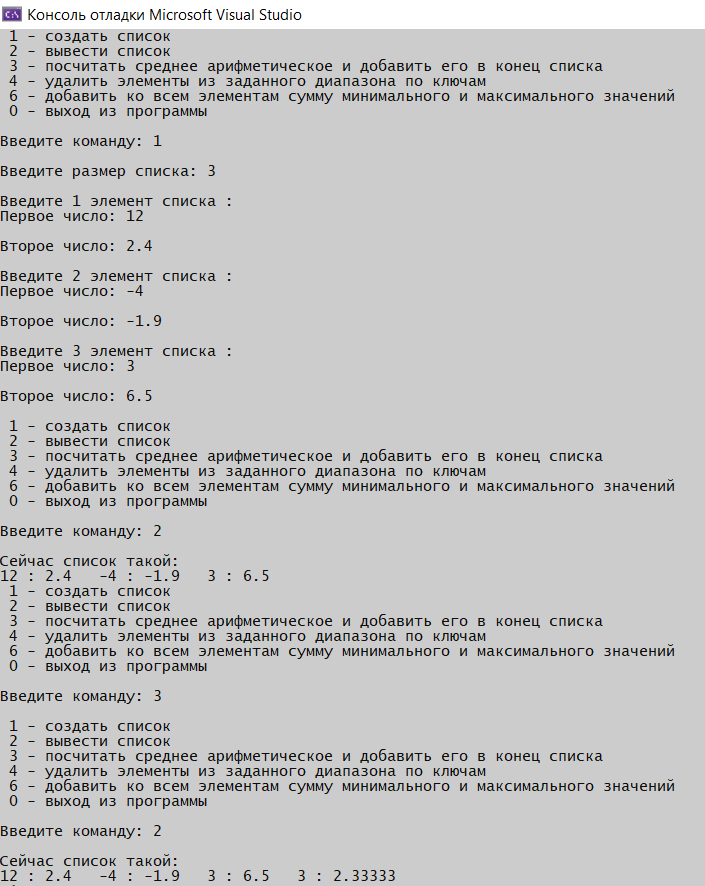
**Скриншоты работы программы**

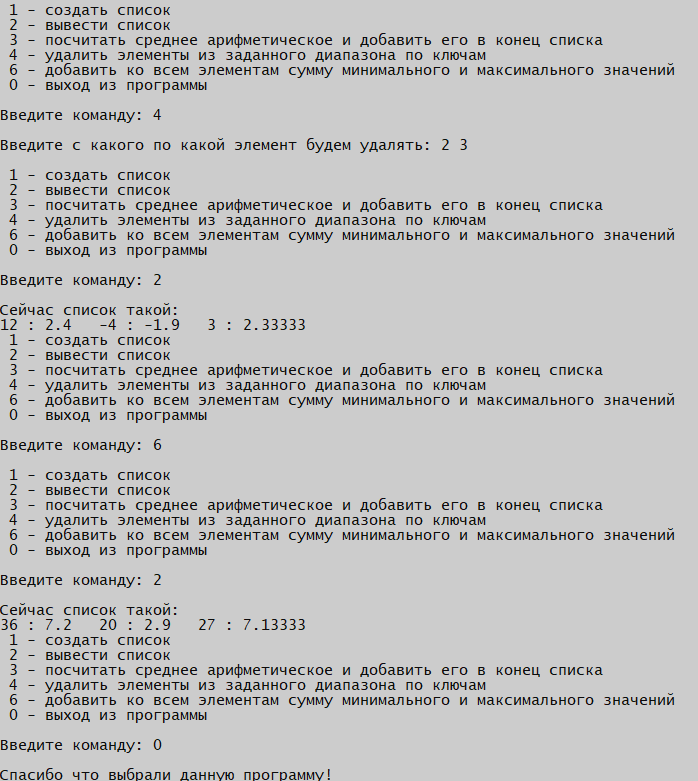
**Задача 1.**

****

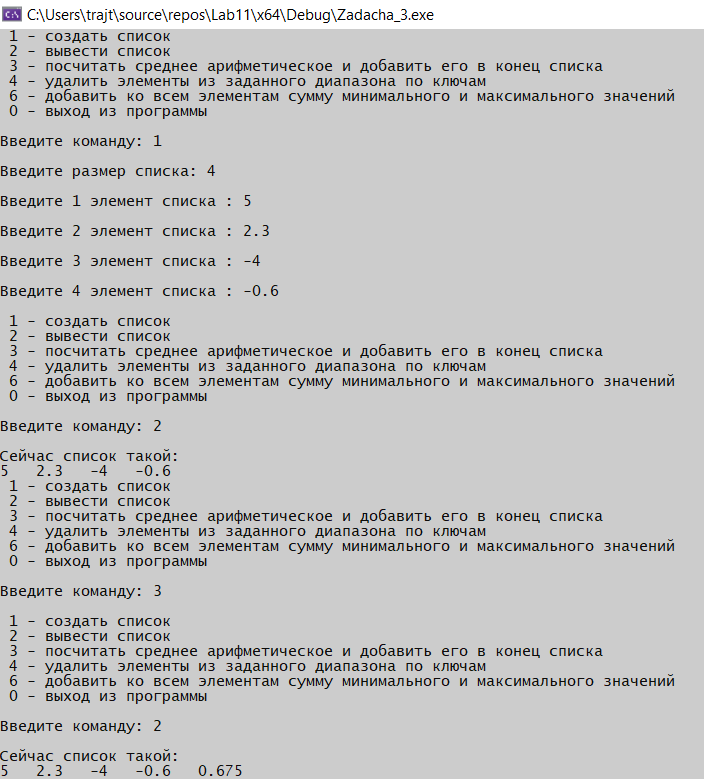
****

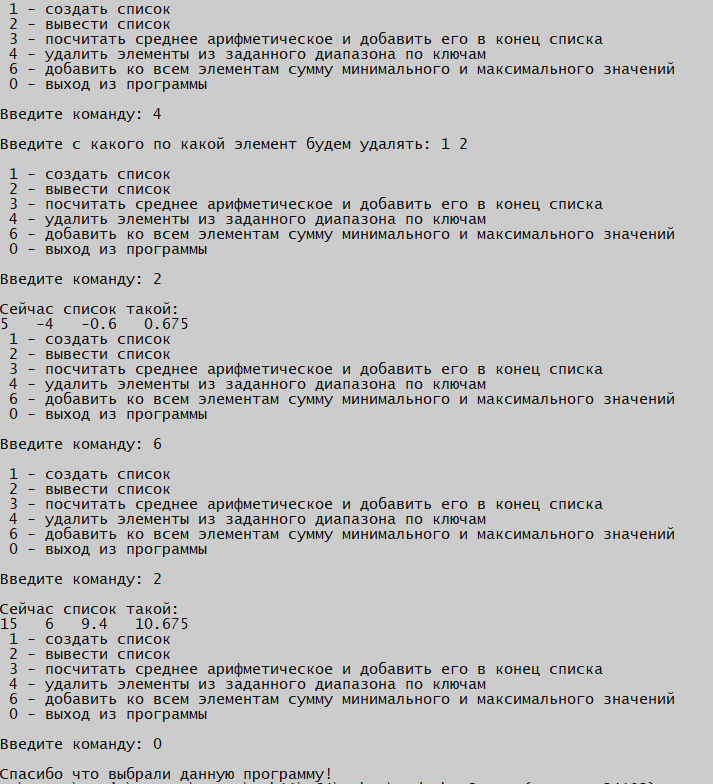
**Задача 2.**

****

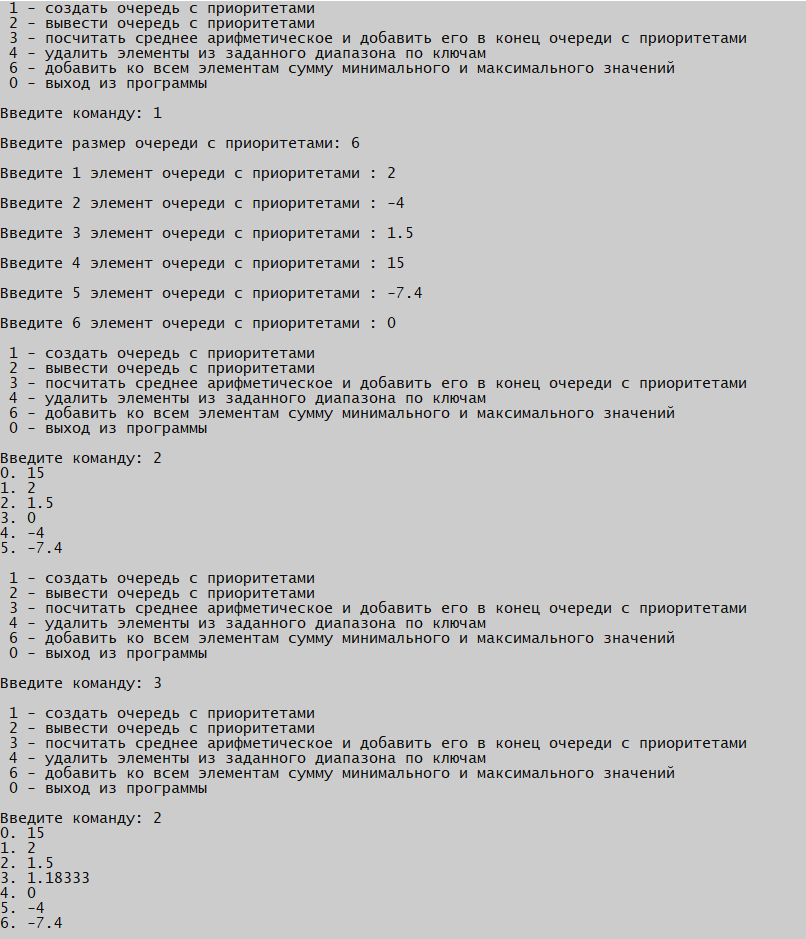
****

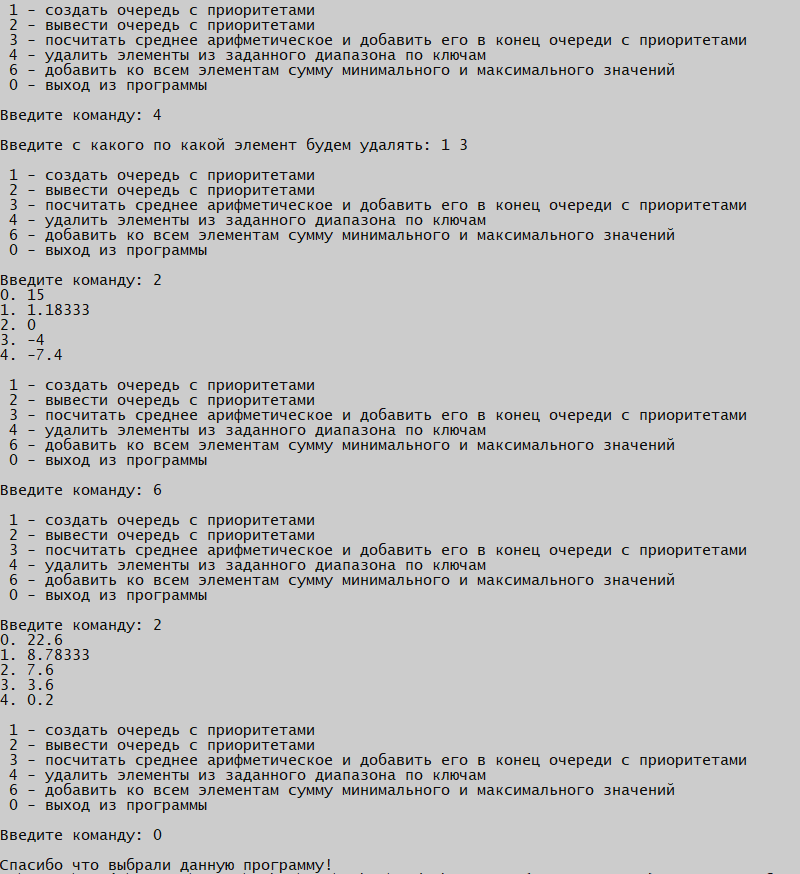
**Задача 3.**

****

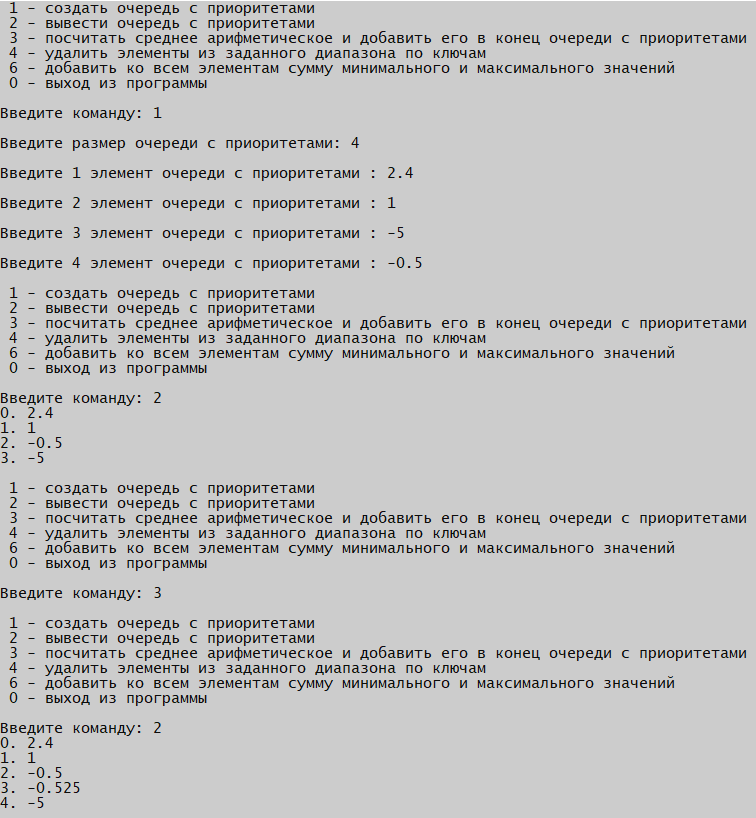
****

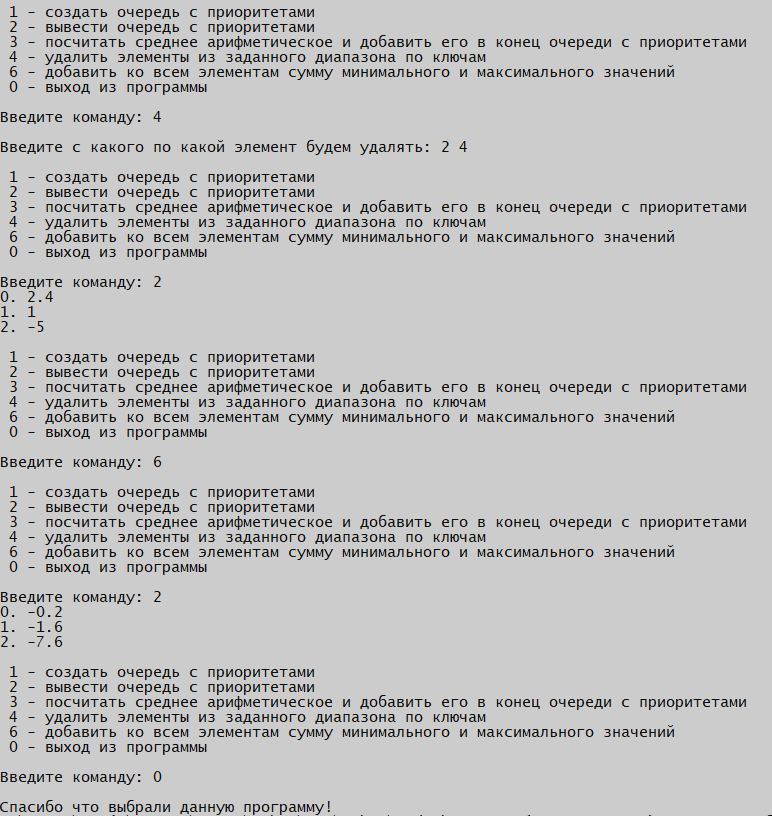
**Задача 4.**

****

****

**Задача 5.**

****

****

**Код программы**

**Задача 1.**

**Lab11\_main.cpp:**

#include <iostream>

#include <list>

#include <cstdlib>

#include <string>

#include <Windows.h>

using namespace std;

typedef list<double>LinkedList; //определяем тип для работы со списком

//функция для формирования списка

LinkedList make\_list(LinkedList l, int n)

{

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

{

double a;

cout << "\nВведите " << i + 1 << " элемент списка : ";

cin >> a;

l.push\_back(a); //добавление a в конец списка

}

return l;

}

//функция для печати списка

void print\_list(LinkedList l)

{

cout << "\nСейчас список такой: \n";

copy(l.begin(), l.end(), ostream\_iterator<double>(cout, " "));

}

//функция для подсчета среднего арифметического и добавление его в конец списка

LinkedList arithmetic\_mean(LinkedList l)

{

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

double s = 0;

//перебор списка

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

{

s += \*it;

it++;

}

double n = l.size(); //количество элементов в векторе

l.push\_back(s/n);

return l;

}

//функция удаления элементов из заданного диапазона по ключам

LinkedList delete\_elem(LinkedList l, int start, int finish)

{

list<double>::iterator it\_start = l.begin();

list<double>::iterator it\_finish = l.begin();

advance(it\_start, start); //смещение итератора до начальной точки отсчета

advance(it\_finish, finish); //смещение итератор до конечной точки отсчета

l.erase(it\_start, it\_finish);

return l;

}

//функция удаления элементов из заданного диапазона по значениям

LinkedList delete\_elem(LinkedList l, int start, int finish, double k)

{

list<double>::iterator it\_start = l.begin();

list<double>::iterator it\_finish = l.begin();

advance(it\_start, start); //смещение итератора от начала до начальной точки отсчета

advance(it\_finish, finish); //смещение итератор от конца до конечной точки отсчета

for (it\_start; it\_start != it\_finish; ++it\_start)

{

l.remove(k); //удаление элементов по значению

}

return l;

}

//функция добавления суммы минимального и максимального значений

LinkedList sum\_min\_max(LinkedList l)

{

list<double> copy\_l;

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

double max = l.front(); //здесь будет максимальное значение в списке

//цикл нахождения максимального значения

for (it; it != l.end(); ++it)

{

if (max < \*it)

max = \*it;

}

double min = l.front(); //здесь будет минимальное значение в списке

//цикл нахождения минимального значения

for (it; it != l.end(); ++it)

{

if (min > \*it)

min = \*it;

}

double s = min + max; //сумма минимального и максимального элементов

//добавление к элементам s

it = l.begin();

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

{

double a = \*it + s;

copy\_l.push\_back(a);

it++;

}

return copy\_l;

}

const char menu[] = " 1 - создать список\n"

" 2 - вывести список\n"

" 3 - посчитать среднее арифметическое и добавить его в конец списка\n"

" 4 - удалить элементы из заданного диапазона по ключам\n"

" 5 - удалить элементы из заданного диапазона по значению\n"

" 6 - добавить ко всем элементам сумму минимального и максимального значений\n"

" 0 - выход из программы\n";

//основная функция

int main()

{

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try

{

list<double> l; //список

int n = 0; //размер списка

for (;;)

{

cout << menu;

int start, finish, k;

int cmd; //команда

cout << "\nВведите команду: ";

cin >> cmd;

switch (cmd)

{

case 1: //создать новый список

l.clear();

cout << "\nВведите размер списка: ";

cin >> n;

l = make\_list(l, n);

break;

case 2: //вывести список

if (n > 0)

print\_list(l);

else

cout << "\nСейчас список пуст";

break;

case 3: //посчитать среднее арифметическое и добавить его в конец списка

l = arithmetic\_mean(l);

break;

case 4: //удалить элементы из заданного диапазона по ключам

cout << "\nВведите с какого по какой элемент будем удалять: ";

cin >> start >> finish;

l = delete\_elem(l, start, finish);

break;

case 5: //удалить элементы из заданного диапазона по значению

cout << "\nВведите с какого по какой элемент будем удалять: ";

cin >> start >> finish;

cout << "\nВведите значение удаляемых элементов: ";

cin >> k;

l = delete\_elem(l, start, finish, k);

break;

case 6: //добавить ко всем элементам сумму минимального и максимального значений

l = sum\_min\_max(l);

break;

case 0: //выход из программы

cout << "\nСпасибо что выбрали данную программу!";

return 0;

default: //неправильно выбранная команда

cout << "\nНеправильная команда, попробуйте другую";

}

cout << endl;

system("pause");

system("cls");

}

}

catch (double) //блок обработки ошибок

{

cout << "Ошибка!";

}

return 0;

}

**Задача 2.**

**Pair.h:**

#pragma once

#pragma once

#include <iostream>

using namespace std;

class **Pair**

{

public:

int a;

double b;

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

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

**Pair**(int a, double b);

**Pair**();

**Pair**(const Pair& p);

~**Pair**();

Pair& operator=(const Pair& p);

Pair operator-(const Pair& p);

Pair& operator+(int a);

Pair& operator+(double b);

Pair& operator++();

Pair operator++(int);

Pair operator+(const Pair& p);

bool operator!=(const Pair& p) { return !((this->a == p.a) && (this->b == p.b)); }

bool operator==(const Pair& p) { return ((this->a == p.a) && (this->b == p.b)); }

bool operator>(const Pair& p) { return (this->a > p.a) && (this->b > p.b); }

bool operator>=(const Pair& p) { return (this->a >= p.a) && (this->b >= p.b); }

bool operator<=(const Pair& p) { return (this->a <= p.a) && (this->b <= p.b); }

bool operator<(const Pair& p) { return (this->a < p.a) && (this->b < p.b); }

Pair operator \* (Pair& pair)

{

Pair new\_pair;

new\_pair.a = this->a \* pair.a;

new\_pair.b = this->b \* pair.b;

return new\_pair;

}

};

**Pair.cpp:**

#include "Pair.h"

Pair::**Pair**(int a, double b)

{

this->a = a;

this->b = b;

}

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

{

cout << "(int)a : ";

in >> p.a;

cout << "(double)b :";

in >> p.b;

return in;

}

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

{

return (out << p.a << " : " << p.b);

}

Pair::**Pair**()

{

a = 0;

b = 0;

}

Pair::~**Pair**()

{

}

Pair::**Pair**(const Pair& p)

{

a = p.a;

b = p.b;

}

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

{

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

a = p.a;

b = p.b;

return \*this;

}

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

{

Pair res(a - p.a, b - p.b);

return res;

}

Pair& Pair::operator+(int a)

{

this->a += a;

return \*this;

}

Pair& Pair::operator+(double b)

{

this->b += b;

return \*this;

}

Pair& Pair::operator++()

{

++a; ++b;

return \*this;

}

Pair Pair::operator ++(int)

{

Pair temp = \*this;

this->a++; this->b++;

return temp;

}

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

{

this->a += p.a;

this->b += p.b;

return \*this;

}

**Zadacha2\_main.cpp:**

#include <iostream>

#include <list>

#include "Pair.h"

#include "LinkedList.h"

#include <queue>

using namespace std;

template <typename T>

void **show\_lst**(list<T>& lst)

{

int i = 1;

for (T it : lst)

{

cout << " " << i++ << ".\t" << it << "\n";

}

}

template <typename T>

void **show\_lst**(LinkedList<T>& lst)

{

int i = 1;

for (T it : lst)

{

cout << " " << i++ << ".\t" << it << "\n";

}

}

void **ex\_1**()

{

list <double> lst;

for (int i = 1; i <= 8; i++)

{

lst.push\_back(i);

}

cout << "??????????? ??????:\n";

show\_lst(*lst*);

double total = 0;

for (double c : lst)

total += c;

lst.push\_back(total / lst.size());

cout << "??????? ????., ??????????? ? ?????:\n";

show\_lst(*lst*);

list <double> tmp = lst;

lst.clear();

int k = 1;

int i, j;

double res;

cout << "????????: "; cin >> i >> j;

cout << "????: "; cin >> res;

for (double c : tmp)

{

if ((k < i) || (k > j) || (c != res))

{

lst.push\_back(c);

}

k++;

}

cout << "????????? ???????? ?? ????????? ?????????:\n";

show\_lst(*lst*);

double max = \*max\_element(lst.begin(), lst.end());

double min = \*min\_element(lst.begin(), lst.end());

for (double& o : lst)

{

o += max + min;

}

cout << "???????? ????. ? ???. ? ??????? ????????:\n";

show\_lst(*lst*);

}

void **ex\_2**()

{

list <Pair> lst;

lst.push\_back(Pair(1, 1.5));

lst.push\_back(Pair(-2, 2.7));

lst.push\_back(Pair(3, -3.2));

cout << "??????????? ??????:\n";

show\_lst(*lst*);

Pair pr(0, 0);

for (Pair c : lst)

{

pr = pr + c;

}

pr.a /= lst.size();

pr.b /= lst.size();

lst.push\_back(pr);

cout << "??????? ????., ??????????? ? ?????:\n";

show\_lst(*lst*);

list <Pair> tmp = lst;

lst.clear();

int k = 1;

int i, j;

Pair res;

cout << "????????: "; cin >> i >> j;

cout << "????: "; cin >> res;

for (Pair c : tmp)

{

if ((k < i) || (k > j) || (c != res))

{

lst.push\_back(c);

}

k++;

}

cout << "????????? ???????? ?? ????????? ?????????:\n";

show\_lst(*lst*);

Pair max = \*max\_element(lst.begin(), lst.end());

Pair min = \*min\_element(lst.begin(), lst.end());

for (Pair& o : lst)

{

o = o + max + min;

}

cout << "???????? ????. ? ???. ? ??????? ????????:\n";

show\_lst(*lst*);

}

template <typename T>

void **tuda\_ego**(LinkedList <T>& lst, LinkedList <T>& tmp, int i, int size)

{

tmp.push(lst.pop());

if (i == size) return;

tuda\_ego(lst, tmp, i + 1, size);

}

void **ex\_3**()

{

LinkedList <Pair> lst;

lst.push(Pair(1, 1.5));

lst.push(Pair(-2, 2.7));

lst.push(Pair(3, -3.2));

cout << "??????????? ??????:\n";

show\_lst(*lst*);

Pair pr(0, 0);

for (Pair c : lst)

{

pr = pr + c;

}

pr.a /= lst();

pr.b /= lst();

lst.push(pr);

cout << "??????? ????., ??????????? ? ?????:\n";

show\_lst(*lst*);

LinkedList <Pair> temp;

tuda\_ego(*lst*, *temp*, 1, lst());

int k = 1;

int i, j;

Pair res;

cout << "????????: "; cin >> i >> j;

cout << "????: "; cin >> res;

for (Pair c : temp)

{

if ((k < i) || (k > j) || (c != res))

{

lst.push(c);

}

k++;

}

cout << "????????? ???????? ?? ????????? ?????????:\n";

show\_lst(*lst*);

Pair max = lst[0];

Pair min = lst[0];

for (auto it = lst.begin(); it != lst.end(); it++)

{

if (\*it > max) max = \*it;

if (\*it < min) min = \*it;

}

for (int count = 0; count < lst(); count++)

{

lst[count] = lst[count] + max + min;

}

cout << "???????? ????. ? ???. ? ??????? ????????:\n";

show\_lst(*lst*);

}

bool operator<(const Pair& pr1, const Pair& pr2)

{

return ((pr1.a < pr2.a) && (pr1.b < pr2.b));

}

void **ex\_4**()

{

priority\_queue <Pair> lst;

priority\_queue <Pair> tmp;

lst.push(Pair(1, 1.5));

lst.push(Pair(-2, 2.7));

lst.push(Pair(3, -3.2));

cout << "??????????? ??????:\n";

Pair sr(0, 0);

while (!lst.empty())

{

cout << lst.top() << endl;

sr = sr + lst.top();

tmp.push(lst.top());

lst.pop();

}

sr.a /= tmp.size();

sr.b /= tmp.size();

cout << "??????? ????., ??????????? ? ?????:\n";

while (!tmp.empty())

{

cout << tmp.top() << endl;

lst.push(tmp.top());

tmp.pop();

}

lst.push(sr);

cout << sr << endl;

int k = 1;

int i, j;

Pair res;

cout << "????????: "; cin >> i >> j;

cout << "????: "; cin >> res;

while (!lst.empty())

{

if ((k < i) || (k > j) || (res != lst.top()))

{

tmp.push(lst.top());

}

lst.pop();

k++;

}

cout << "????????? ???????? ?? ????????? ?????????:\n";

Pair max = tmp.top();

Pair min = tmp.top();

while (!tmp.empty())

{

if (max < tmp.top()) max = tmp.top();

if (tmp.top() < min) min = tmp.top();

cout << tmp.top() << endl;

lst.push(tmp.top());

tmp.pop();

}

//3

cout << "???????? ????. ? ???. ? ??????? ????????:\n";

while (!lst.empty())

{

tmp.push(max + min + lst.top());

lst.pop();

}

while (!tmp.empty())

{

lst.push(tmp.top());

cout << tmp.top() << endl;

tmp.pop();

}

}

void **ex\_5**()

{

ex\_3();

}

int **main**()

{

setlocale(0, ".1251");

ex\_1();

cout << "////////////" << endl;

ex\_2();

cout << "////////////" << endl;

ex\_3();

cout << "////////////" << endl;

ex\_4();

cout << "////////////" << endl;

ex\_5();

return 0;

}

**Задача 3.**

**LinkedList.h:**

#pragma once

#pragma once

#include <iostream>

#include <algorithm>

#include "D:\?++\Lab\_OOP\_9\Error.h"

using namespace std;

template<typename T>

struct **Point**

{

T num;

Point\* next = nullptr;

};

template <typename T>

class **Iterator**

{

private:

Point<T>\* elem = nullptr;

public:

**Iterator**() { elem = nullptr; }

**Iterator**(Point<T>\* point) { elem = point; }

**Iterator**(const Iterator& it) { elem = it.elem; };

bool operator == (const Iterator& it) { return elem == it.elem; }

bool operator != (const Iterator& it) { return elem != it.elem; }

Iterator& operator ++ ();

Iterator operator ++ (int);

Iterator& operator + (int n);

T operator \* () const;

};

template <typename T>

T Iterator<T>::operator \* () const

{

if (elem)

return elem->num;

else

throw IndexError2();

}

template <typename T>

Iterator<T>& Iterator<T>::operator ++ ()

{

if (elem == nullptr) throw IndexError2();

elem = elem->next;

return \*this;

}

template <typename T>

Iterator<T> Iterator<T>::operator ++ (int)

{

if (elem == nullptr) throw IndexError2();

Iterator<T> temp(elem);

++(\*this);

return temp;

}

template <typename T>

Iterator<T>& Iterator<T>::operator + (int n)

{

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

{

if (elem == nullptr) throw IndexError2();

elem = elem->next;

}

return \*this;

}

//-------------------------//

// ????????? ??????

//-------------------------//

template <typename T>

class **LinkedList**

{

private:

size\_t size = 0;

Point<T>\* head = nullptr;

Point<T>\* top = nullptr;

void **output**(Point<T>\* obj) const;

void **show**() const;

public:

friend ostream& operator<<(ostream& out, const LinkedList<T>& list)

{

list.show();

return out;

}

friend istream& operator>>(istream& in, LinkedList<T>& list)

{

T k;

in >> k;

list.push(k);

return in;

}

**LinkedList**();

**LinkedList**(size\_t s, T k);

~**LinkedList**();

**LinkedList**(const LinkedList<T>& list);

T& operator [](int index) const;

LinkedList& operator =(const LinkedList<T>& list);

LinkedList operator \* (LinkedList<T>& list);

int operator ()() { return size; };

void **push**(T k);

T **pop**();

Iterator<T> **begin**() { return Iterator<T>(head); }

Iterator<T> **end**() { return Iterator<T>(top->next); }

};

template <typename T>

LinkedList<T>::**LinkedList**()

{

head = nullptr;

top = nullptr;

size = 0;

}

template <typename T>

LinkedList<T>::**LinkedList**(size\_t s, T k)

{

size = s;

if (size != 0) {

Point<T>\* obj = new Point<T>;

obj->num = k;

head = obj;

Point<T>\* temp = head;

for (int i = size - 1; i != 0; i--)

{

obj = new Point<T>;

temp->next = obj;

obj->num = k;

temp = obj;

}

top = obj;

}

}

template <typename T>

void LinkedList<T>::**push**(T k)

{

if (size == 0)

{

head = new Point<T>;

head->num = k;

top = head;

}

else {

Point<T>\* obj = new Point<T>;

top->next = obj;

obj->num = k;

top = obj;

}

size++;

}

template <typename T>

T LinkedList<T>::**pop**()

{

if (size == 0) throw EmptySizeError();

T k = head->num;

Point<T>\* temp = head;

head = head->next;

size--;

delete temp;

return k;

}

template <typename T>

LinkedList<T>::~**LinkedList**()

{

cout << endl << "?????? ??????????!" << endl << endl;

while (head != nullptr)

{

pop();

}

}

template <typename T>

void LinkedList<T>::**output**(Point<T>\* obj) const

{

cout << obj->num << " ";

if (obj == top) return;

output(*obj->*next);

}

template <typename T>

void LinkedList<T>::**show**() const

{

if (size == 0) cout << "?????? ????" << endl;

else output(*head*);

}

template <typename T>

T& LinkedList<T>::operator[](int index) const

{

if (index == 0 && size == 0) throw IndexError();

if (index < 0) throw IndexError1();

if (index >= size) throw IndexError2();

Point<T>\* temp = head;

int count = 0;

while (temp != nullptr)

{

if (count == index)

return temp->num;

temp = temp->next;

++count;

}

}

template <typename T>

LinkedList<T>::**LinkedList**(const LinkedList<T>& list)

{

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

{

this->push(list[i]);

}

}

template <typename T>

LinkedList<T>& LinkedList<T>::operator =(const LinkedList<T>& list)

{

int size\_temp = size;

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

{

this->pop();

}

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

{

this->push(list[i]);

}

return \*this;

}

template <typename T>

LinkedList<T> LinkedList<T>::operator \* (LinkedList<T>& list)

{

T nulik;

nulik = 0;

LinkedList<T> new\_list;

for (int i = 0; i < min((int)size, list()); i++)

{

new\_list.push((\*this)[i] \* list[i]);

}

for (int i = 0; i < max((int)size, list()) - min((int)size, list()); i++)

{

new\_list.push(nulik);

}

return new\_list;