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:

Студенты группы РИС-22-2б

Малых В.О.

Проверил доц. Кафедры ИТАС:

Полякова О.А.

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.

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. Перечислить операции и методы общие для всех контейнеров.

|  |  |
| --- | --- |
| **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. 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 vector.

|  |  |
| --- | --- |
| Vector | |
| push\_back() | добавление  в конец |
| pop\_back() | удаление из конца |
| insert | Вставка в  произвольное место |
| erase | удаление из произвольного места |
| []  at | доступ к произвольному элементу |
| clear() | очистить  вектор |
| swap | Обмен векторов |

1. Перечислить методы, которые поддерживает последовательный контейнер list.

|  |  |
| --- | --- |
| 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 произвольное место |
| erase | удаление из произвольного места |
| clear() | Очистить вектор |
| swap | Обмен списков |
| splice | Сцепка списков |

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)

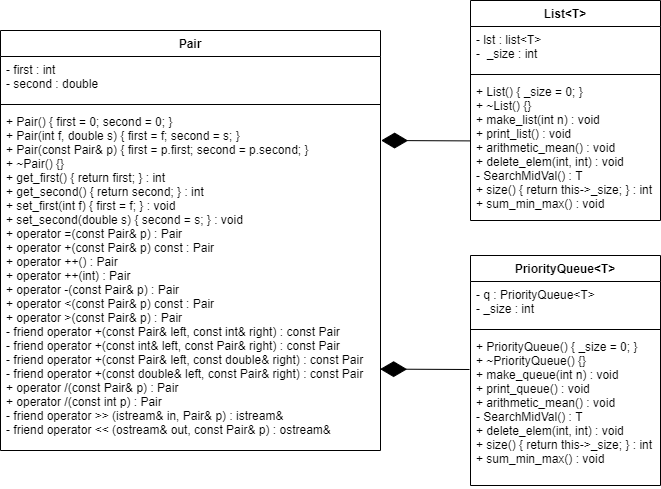
{

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

cout << \*i << endl;

}

**UML таблица**

****

**Код программы**

**Задача 1.**

**Main1.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

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

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

Pair(int f, double s) { first = f; second = s; }

Pair(const Pair& p) { first = p.first; second = p.second; }

~Pair() {};

int get\_first() { return first; }

int get\_second() { return second; }

void set\_first(int f) { first = f; }

void set\_second(double s) { second = s; }

//перегруженные операции

Pair& operator =(const Pair& p);

Pair operator +(const Pair& p) const;

Pair& operator ++();

Pair operator ++(int);

Pair operator -(const Pair& p);

bool operator <(const Pair& p) const;

bool operator >(const Pair& p);

friend const Pair operator +(const Pair& left, const int& right);

friend const Pair operator +(const int& left, const Pair& right);

friend const Pair operator +(const Pair& left, const double& right);

friend const Pair operator +(const double& left, const Pair& right);

Pair operator /(const Pair& p);

Pair operator /(const int p);

//глобальные функции ввода-вывода

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;

//перегрузка операции присваивания

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

{

//Проверка на самоприсваивание

if (&p == this)

return \*this;

else

first = p.first;

second = p.second;

return \*this;

}

//перегрузка операции сложение

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

{

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

return temp;

}

//перегрузка инкремента

Pair& Pair::operator++()

{

++this->first;

++this->second;

return \*this;

}

Pair Pair::operator++(int)

{

Pair temp(\*this);

++this->first;

++this->second;

return temp;

}

//перегрузка операции вычитание

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

{

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

}

const Pair operator+(const Pair& left, const int& right)

{

return Pair(left.first + right, left.second);

}

const Pair operator+(const int& left, const Pair& right)

{

return right + left;

}

const Pair operator+(const Pair& left, const double& right)

{

return Pair(left.first, left.second + right);

}

const Pair operator+(const double& left, const Pair& right)

{

return right + left;

}

//перегрузка операции деление

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

{

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

}

Pair Pair::operator/(const int p)

{

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

}

//перегрузка операции сравнивания

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

{

double thiss = this->first + this->second, otherr = p.first + p.second;

if (thiss < otherr) return true;

else return false;

}

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

{

return !(\*this < p);

}

//перегрузка глобальной функции-операции ввода

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

{

cout << "\nПервое число: ";

in >> p.first;

cout << "\nВторое число: ";

in >> p.second;

return in;

}

//перегрузка глобальной функции-операции вывода

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

{

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

}

**Main2.cpp:**

#include <iostream>

#include <list>

#include <cstdlib>

#include <string>

#include <Windows.h>

#include "Pair.h"

using namespace std;

typedef list<Pair>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++)

{

Pair 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<Pair>(cout, " "));

}

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

LinkedList arithmetic\_mean(LinkedList l)

{

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

Pair s;

//перебор списка

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

{

s = s + \*it;

it++;

}

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

list<Pair>::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, Pair k)

//{

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

// list<Pair>::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<Pair> copy\_l;

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

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

//цикл нахождения максимального значения

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

{

if (max < \*it)

max = \*it;

}

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

//цикл нахождения минимального значения

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

{

if (min > \*it)

min = \*it;

}

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

//добавление к элементам s

it = l.begin();

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

{

Pair 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"

" 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<Pair> l; //список

int n = 0; //размер списка

for (;;)

{

cout << menu;

int start, finish;

/\*Pair 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 3.**

**LinkedList.h:**

#pragma once

#include <iostream>

#include <list>

using namespace std;

//класс списка

template <typename T>

class List

{

private:

list<T> lst;

T SearchMidVal();

int \_size;

public:

List() { \_size = 0; }

~List() {};

void make\_list(int n);

void print\_list();

void arithmetic\_mean();

void delete\_elem(int, int);

int size() { return this->\_size; }

void sum\_min\_max();

};

//создание списка

template<class T>

void List<T>::make\_list(int n)

{

T a;

for (size\_t i = 0; i < n; i++)

{

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

cin >> a;

this->lst.push\_back(a);

}

\_size = n;

}

//вывод списка

template<class T>

void List<T>::print\_list()

{

cout << "\nСейчас список такой: \n";

copy(lst.begin(), lst.end(), ostream\_iterator<T>(cout, " "));

}

template<class T>

T List<T>::SearchMidVal()

{

T mid = 0;

for (auto& i : lst)

mid = mid + i;

return mid / lst.size();

}

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

template<typename T>

void List<T>::arithmetic\_mean()

{

lst.push\_back(SearchMidVal());

this->\_size = lst.size();

}

//функция удаления элементов из заданного диапазона

template<typename T>

void List<T>::delete\_elem(int start, int finish)

{

auto it\_start = lst.begin();

auto it\_finish = lst.begin();

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

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

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

}

//функция добавления суммы минимального и максимального значений

template<typename T>

void List<T>::sum\_min\_max()

{

list<double> copy\_l;

auto it = lst.begin();

T max = lst.front(); //здесь будет максимальное значение в списке

//цикл нахождения максимального значения

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

{

if (max < \*it)

max = \*it;

}

T min = lst.front(); //здесь будет минимальное значение в списке

//цикл нахождения минимального значения

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

{

if (min > \*it)

min = \*it;

}

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

//добавление к элементам s

it = lst.begin();

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

{

T a = \*it + s;

copy\_l.push\_back(a);

it++;

}

lst = copy\_l;

}

**Main3.cpp:**

#include <iostream>

#include <list>

#include <cstdlib>

#include <string>

#include <Windows.h>

#include "LinkedList.h"

using namespace std;

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

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"

" 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; //размер списка

for (;;)

{

cout << menu;

int start, finish;

int cmd; //команда

cout << "\nВведите команду: ";

cin >> cmd;

switch (cmd)

{

case 1: //создать новый список

cout << "\nВведите размер списка: ";

cin >> n;

l.make\_list(n);

break;

case 2: //вывести список

if (n > 0)

l.print\_list();

else

cout << "\nСейчас список пуст";

break;

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

l.arithmetic\_mean();

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(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();

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 4.**

**Main4.cpp:**

#include <C:\Users\vovam\source\repos\L11\_OOP\L11\_OOP\Pair.h>

#include <list>

#include <queue>

#include <cstdlib>

#include <Windows.h>

typedef priority\_queue<Pair>PQ; //очередь с приоритетами

typedef list<Pair>LinkedList; //список

//функция для создания очереди с приоритетами

PQ make\_queue(PQ s, int n)

{

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

{

Pair a;

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

cin >> a;

s.push(a); //добавление a в конец очереди

}

return s;

}

//функция для печати очереди с приоритетами

void print\_queue(PQ q)

{

int j = 0;

cout << "\nСейчас список такой: \n";

//priority\_queue<Pair> temp(q);

while (!q.empty())

{

cout << j++ << ". " << q.top() << endl;

q.pop();

}

}

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

PQ arithmetic\_mean(PQ q)

{

priority\_queue<Pair> temp(q);

Pair s;

while (!temp.empty())

{

s = s + temp.top();

temp.pop();

}

q.push( s / q.size() );

return q;

}

//функция удаления элементов из заданного диапазона по ключам

PQ delete\_elem(PQ q, int start, int finish)

{

priority\_queue<Pair> temp;

Pair t;

int n = 1;

for(n; n <= start; n++)

{

t = q.top();

q.pop();

temp.push(t);

}

for (n; n <= finish; n++)

{

q.pop();

}

for (n; n <= q.size(); n++)

{

t = q.top();

q.pop();

temp.push(t);

}

return temp;

}

//функция удаления элементов из заданного диапазона по значениям

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

//{

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

// list<Pair>::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

PQ sum\_min\_max(PQ q)

{

priority\_queue<Pair> temp;

Pair t;

Pair max = q.top(); //здесь будет максимальное значение в списке

//цикл нахождения максимального значения

for (int i = 1; i <= q.size(); i++)

{

if (max < q.top())

max = q.top();

}

Pair min = q.top(); //здесь будет минимальное значение в очереди

//цикл нахождения минимального значения

for (int i = 1; i <= q.size(); i++)

{

if (min > q.top())

min = q.top();

}

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

//добавление к элементам s

for (int i = 1; i <= q.size(); i++)

{

Pair a = q.top() + s;

temp.push(a);

}

return temp;

}

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"

" 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<Pair> l; //список

priority\_queue<Pair> s; //очередь с приоритетом

int n = 0; //размер списка

for (;;)

{

cout << menu;

int start, finish;

/\*Pair k;\*/

int cmd; //команда

cout << "\nВведите команду: ";

cin >> cmd;

switch (cmd)

{

case 1: //создать новый список

cout << "\nВведите размер очереди с приоритетом: ";

cin >> n;

s = make\_queue(s, n);

break;

case 2: //вывести список

if (n > 0)

print\_queue(s);

else

cout << "\nСейчас очередь с приоритетом пустая";

break;

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

s = arithmetic\_mean(s);

break;

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

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

cin >> start >> finish;

s = delete\_elem(s, 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

s = sum\_min\_max(s);

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 5.**

**Priority\_queue.h:**

#pragma once

#include <iostream>

#include <list>

#include <queue>

using namespace std;

//класс очереди с приоритетами

template <typename T>

class PriorityQueue

{

private:

priority\_queue<T> q;

T SearchMidVal();

int \_size;

public:

PriorityQueue() { \_size = 0; };

~PriorityQueue() {};

void make\_queue(int n);

void print\_queue();

void arithmetic\_mean();

void delete\_elem(int, int);

int size() { return this->\_size; };

void sum\_min\_max();

};

//создание очереди

template<class T>

void PriorityQueue<T>::make\_queue(int n)

{

T a;

for (size\_t i = 0; i < n; i++)

{

cout << "\nВведите " << i + 1 << " элемент очереди с приоритетами : ";

cin >> a;

this->q.push(a);

}

\_size = n;

}

//вывод очереди

template<class T>

void PriorityQueue<T>::print\_queue()

{

int j = 0;

priority\_queue<double> temp(this->q);

while (!temp.empty())

{

cout << j++ << ". " << temp.top() << endl;

temp.pop();

}

}

template<class T>

T PriorityQueue<T>::SearchMidVal()

{

priority\_queue<T> temp(q);

T mid = 0;

while (!temp.empty())

{

mid = mid + temp.top();

temp.pop();

}

return mid / q.size();

}

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

template<typename T>

void PriorityQueue<T>::arithmetic\_mean()

{

q.push(SearchMidVal());

this->\_size = q.size();

}

//функция удаления элементов из заданного диапазона

template<typename T>

void PriorityQueue<T>::delete\_elem(int start, int finish)

{

priority\_queue<T> temp;

double t;

int n = 1;

int z = q.size();

for (n; n <= start; n++)

{

t = q.top();

q.pop();

temp.push(t);

}

for (n; n <= finish; n++)

{

q.pop();

}

for (n; n <= z; n++)

{

t = q.top();

q.pop();

temp.push(t);

}

q = temp;

}

//функция добавления суммы минимального и максимального значений

template<typename T>

void PriorityQueue<T>::sum\_min\_max()

{

T k;

T loc\_min = q.top(), loc\_max = 0;

priority\_queue<T> temp(q);

while (!temp.empty())

{

k = temp.top();

if (loc\_max < k)

{

loc\_max = k;

temp.pop();

}

else if (loc\_min > k)

{

loc\_min = k;

temp.pop();

}

else temp.pop();

}

temp.swap(q);

while (!temp.empty())

{

k = temp.top();

q.push(k + loc\_min + loc\_max);

temp.pop();

}

}

**Main5.cpp:**

#include <C:\Users\vovam\source\repos\L11\_OOP\L11\_OOP\Pair.h>

#include <iostream>

#include <list>

#include <cstdlib>

#include <string>

#include <Windows.h>

#include "Priority\_queue.h"

using namespace std;

typedef PriorityQueue<double>PQ; //определяем тип для работы со списком

const char menu[] = " 1 - создать очередь с приоритетами\n"

" 2 - вывести очередь с приоритетами\n"

" 3 - посчитать среднее арифметическое и добавить его в конец очереди с приоритетами\n"

" 4 - удалить элементы из заданного диапазона по ключам\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

{

PriorityQueue<double> q; //список

int n; //размер списка

for (;;)

{

cout << menu;

int start, finish;

int cmd; //команда

cout << "\nВведите команду: ";

cin >> cmd;

switch (cmd)

{

case 1: //создать новую очередь с приоритетами

cout << "\nВведите размер очереди с приоритетами: ";

cin >> n;

q.make\_queue(n);

break;

case 2: //вывести очередь с приоритетами

if (n > 0)

q.print\_queue();

else

cout << "\nСейчас очередь с приоритетами пустая";

break;

case 3: //посчитать среднее арифметическое и добавить его в конец очереди с приоритетами

q.arithmetic\_mean();

break;

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

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

cin >> start >> finish;

q.delete\_elem(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

q.sum\_min\_max();

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;

}