**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №18.11 на тему

«Последовательные контейнеры библиотеки STL»

Вариант №11

Выполнил студент группы ИВТ-20-2б

Сабуров Павел Алексеевич

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с последовательными контейнерами в библиотеке STL входящей в поставку с языком программирования C++;

**Постановка задачи**

Задача – реализовать на языке программирования C++ программу, которая демонстрирует возможности работы с последовательными контейнерами в библиотеке STL

**Исходные данные для варианта №11:**

Задание 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.

Задание 2:

1.  Контейнер - вектор

2.  Тип элементов - float

Задание 3:

Найти минимальный элемент и добавить его на заданную позицию контейнера

Задание 4:

Найти элементы большие среднего арифметического и удалить их из контейнера

Задание 5:

Каждый элемент домножать на максимальный элемент контейнера

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code

STL – Standard Template Library, стандартная библиотека шаблонов, которая состоит из двух основных частей:

1. Набор контейнерных классов;
2. Набор обобщённых алгоритмов;

Контейнер – объект, который содержит другие однотипные объекты;

Обобщённый алгоритм – алгоритмы применимые к любым типам данных, возможны благодаря использованию шаблонов;

Итератор – более абстрактное понятие указателя, ссылается на некоторый элемент контейнера;

**Задание №1:**

Функции для обработки контейнера объявляются в заголовочном файле следующим образом:

#pragma once

#include <vector>

#include <iostream>

using namespace std;

vector<float>::iterator IteratorByIndex(vector<float>& processingVector, int index);

float RandomFloat(float minValue, float maxValue);

void CreateRandomVector(vector<float>& processingVector, int count);

void PrintVector(vector<float>& processingVector);

void FindAverageAndPushFront(vector<float>& processingVector);

void FindAndDelete(vector<float>& processingVector, float searchKey);

void SubtractByMinValue(vector<float>& processingVector);

Объявлены следующие функции:

* Функция, которая возвращает случайное число с плавающей запятой;
* Функция, которая создаёт контейнер с случайными элементами;
* Функция вывода контейнера;
* Функция, которая возвращает итератор по индексу элемента;
* Функции, выполняющие основные задания:
  + Найти среднее арифметическое и добавить его в начало;
  + Найти элемент по ключу и удалить его;
  + Вычесть из каждого элемента минимальный элемент контейнера;

Нахождение итератора по индексу:

vector<float>::iterator IteratorByIndex(vector<float>& processingVector, int index)

{

vector<float>::iterator iterator = processingVector.begin();

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

iterator++;

return iterator;

}

Случайное число с плавающей запятой:

float RandomFloat(float minValue, float maxValue)

{

return minValue + static\_cast <float> (rand()) /

(static\_cast <float>(RAND\_MAX / (maxValue-minValue)));

}

Формирование случайного контейнера:

void CreateRandomVector(vector<float>& processingVector, int count)

{

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

processingVector.push\_back(RandomFloat(0, 100));

}

Вывод контейнера на экран:

void PrintVector(vector<float>& processingVector)

{

cout << "\nVector count = " << processingVector.size() << ":\n";

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

cout << "#" << i + 1 << ": " << processingVector[i] << ";\n";

}

Нахождение среднего арифметического и добавление в начало:

void FindAverageAndPushFront(vector<float>& processingVector)

{

float summ = 0;

vector<float> temporary;

temporary.push\_back(0);

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

{

summ += processingVector[i];

temporary.push\_back(processingVector[i]);

}

float average = summ / processingVector.size();

cout << "\nThe average = " << average << ";\n";

temporary[0] = average;

processingVector = temporary;

}

Нахождение элемента по ключу и его удаление:

void FindAndDelete(vector<float>& processingVector, float searchKey)

{

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

if (abs(processingVector[i] - searchKey) <= 0.001)

{

cout << "\nThe key has been founded: index = " << i + 1 << ";\n";

processingVector.erase(IteratorByIndex(processingVector, i));

break;

}

}

Вычитание почленно минимального элемента контейнера:

void SubtractByMinValue(vector<float>& processingVector)

{

float minValue = processingVector[0];

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

if (processingVector[i] < minValue)

minValue = processingVector[i];

cout << "\nMin value = " << minValue << ";\n";

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

processingVector[i] -= minValue;

}

Основная программа, которая запускает все вышеописанные функции:

#include "VectorProcessorFloat.h"

using namespace std;

int main()

{

vector<float> newVector;

CreateRandomVector(newVector, 18);

PrintVector(newVector);

FindAverageAndPushFront(newVector);

PrintVector(newVector);

cout << "\nInput the search key: ";

float key; cin >> key;

FindAndDelete(newVector, key);

PrintVector(newVector);

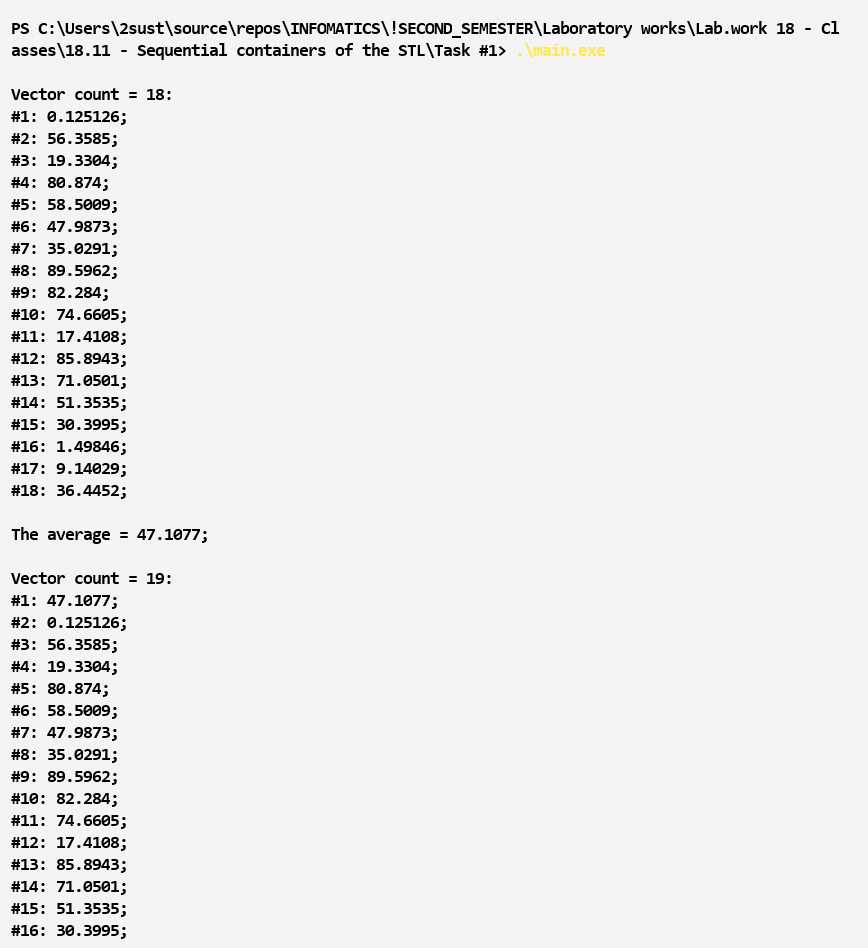
SubtractByMinValue(newVector);

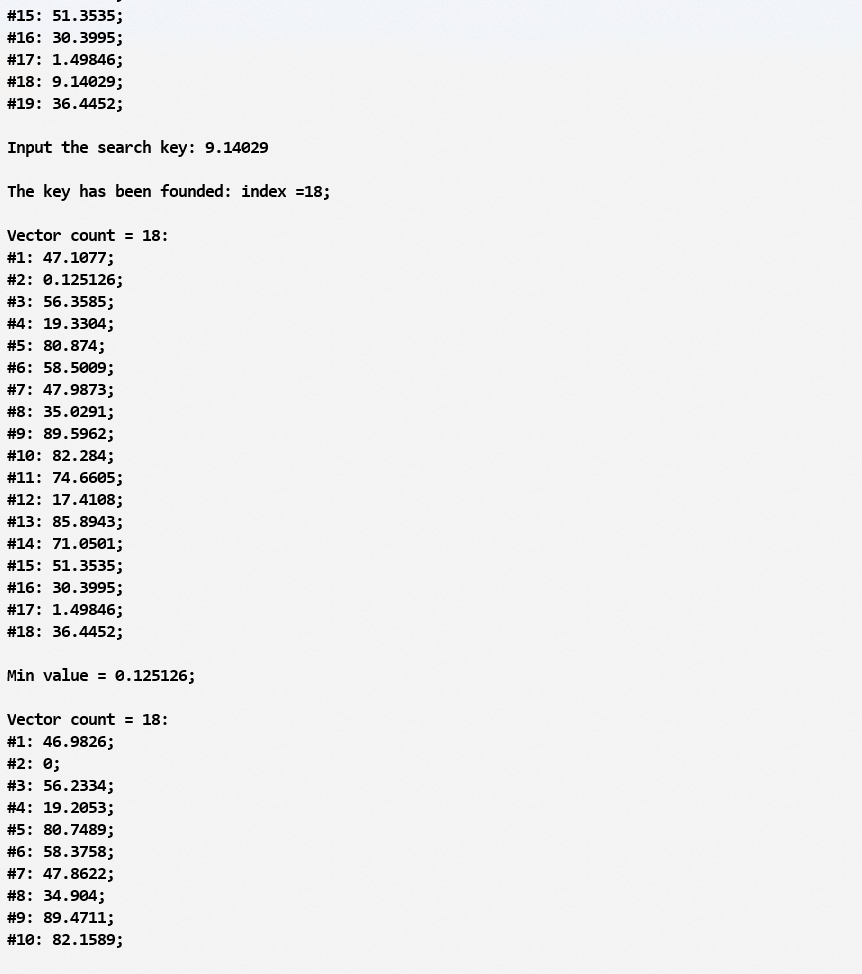
PrintVector(newVector);

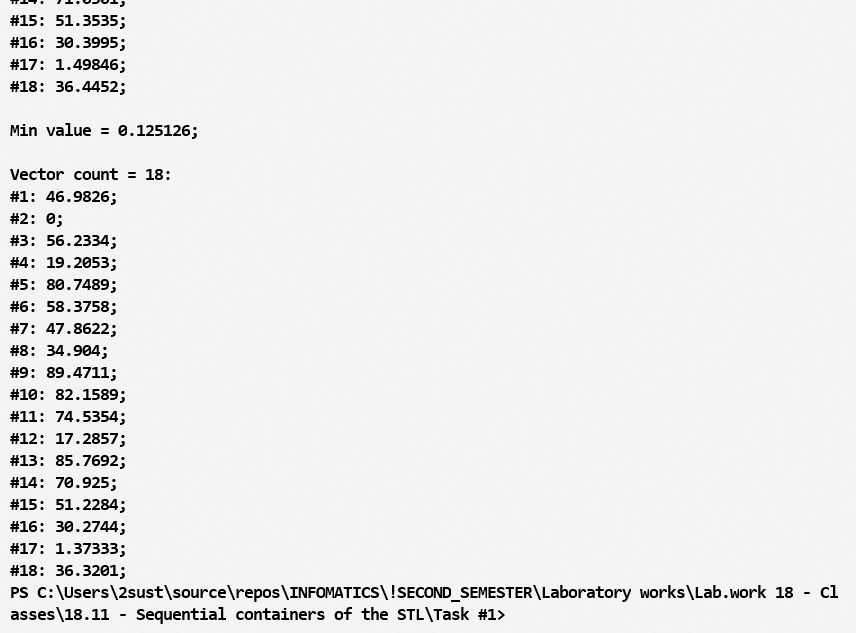
return 0;

}

Скриншоты выполненной программы:







**Задание №2:**

Алгоритмы те же, но вместо стандартного типа заводится пользовательский, в данном случае тип «Деньги» из предыдущих лабораторных работ.

Он содержит следующий заголовочный файл:

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

class Money

{

private:

long \_dollars;

int \_cents;

public:

int Dollars() { return \_dollars; }

void SetDollars(long dollars);

int Cents() { return \_cents; }

void SetCents(int cents);

void PrintMoney();

friend bool operator > (Money& first, Money& second);

friend bool operator < (Money& first, Money& second);

friend bool operator == (Money& first, Money& second);

friend istream& operator >> (istream& in, Money& money);

friend ostream& operator << (ostream& out, Money& money);

friend fstream& operator >> (fstream& fin, Money& money);

friend fstream& operator << (fstream& fout, Money& money);

Money& operator++ ();

Money operator++ (int);

Money& operator = (const Money& value);

Money& operator += (Money& adder);

Money& operator -= (Money& subtractor);

Money& operator / (Money& divider);

Money();

Money(long dollars, int cents);

Money(const Money& parentMoney);

~Money();

};

Поскольку необходимые операторы перегружены, достаточно лишь в заголовочном и ресурсном файле поменять тип float на Money.

Заголовочный файл с функциями примет следующий вид:

#pragma once

#include <vector>

#include <iostream>

#include "Money.h"

using namespace std;

vector<Money>::iterator IteratorByIndex(vector<Money>& processingVector, int index);

Money RandomMoney();

void CreateRandomVector(vector<Money>& processingVector, int count);

void PrintVector(vector<Money>& processingVector);

void FindAverageAndPushFront(vector<Money>& processingVector);

void FindAndDelete(vector<Money>& processingVector, Money searchKey);

void SubtractByMinValue(vector<Money>& processingVector);

Также малые изменение в главной программе:

#include "VectorProcessorMoney.h"

#include "Money.h"

#include <vector>

using namespace std;

int main()

{

vector<Money> newVector;

cout << "Printing new vector:" << endl;

CreateRandomVector(newVector, 18);

PrintVector(newVector);

cout << "Adding new vector:" << endl;

FindAverageAndPushFront(newVector);

PrintVector(newVector);

cout << "\nInput the search key:\n";

Money key; cin >> key;

FindAndDelete(newVector, key);

PrintVector(newVector);

cout << "Subtracting money by min value";

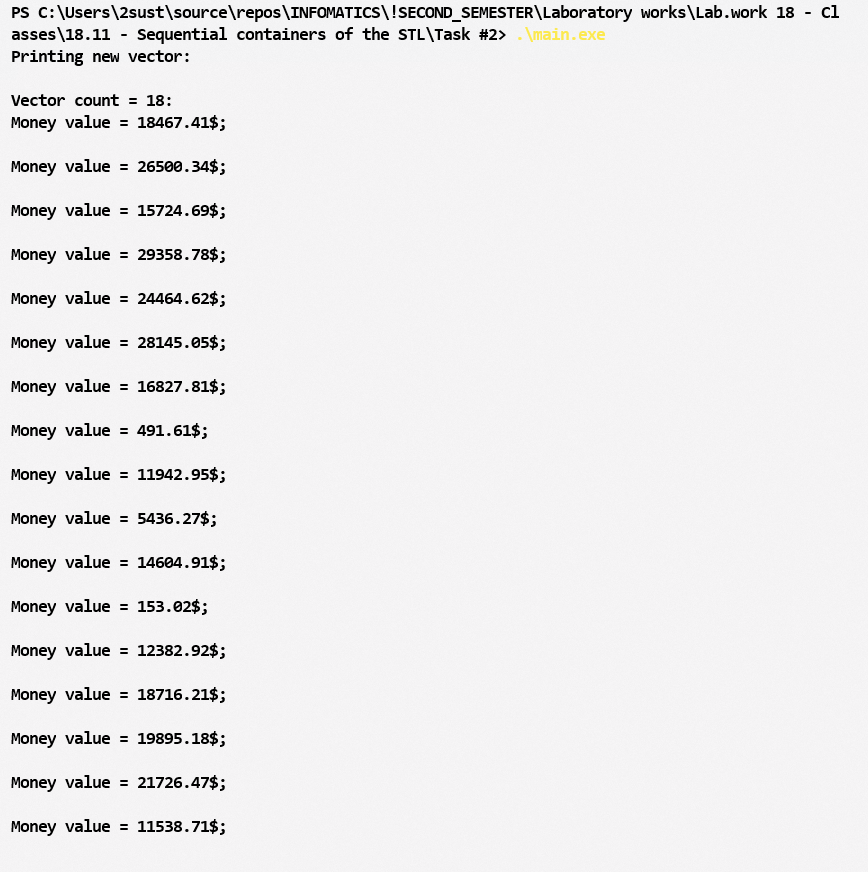
SubtractByMinValue(newVector);

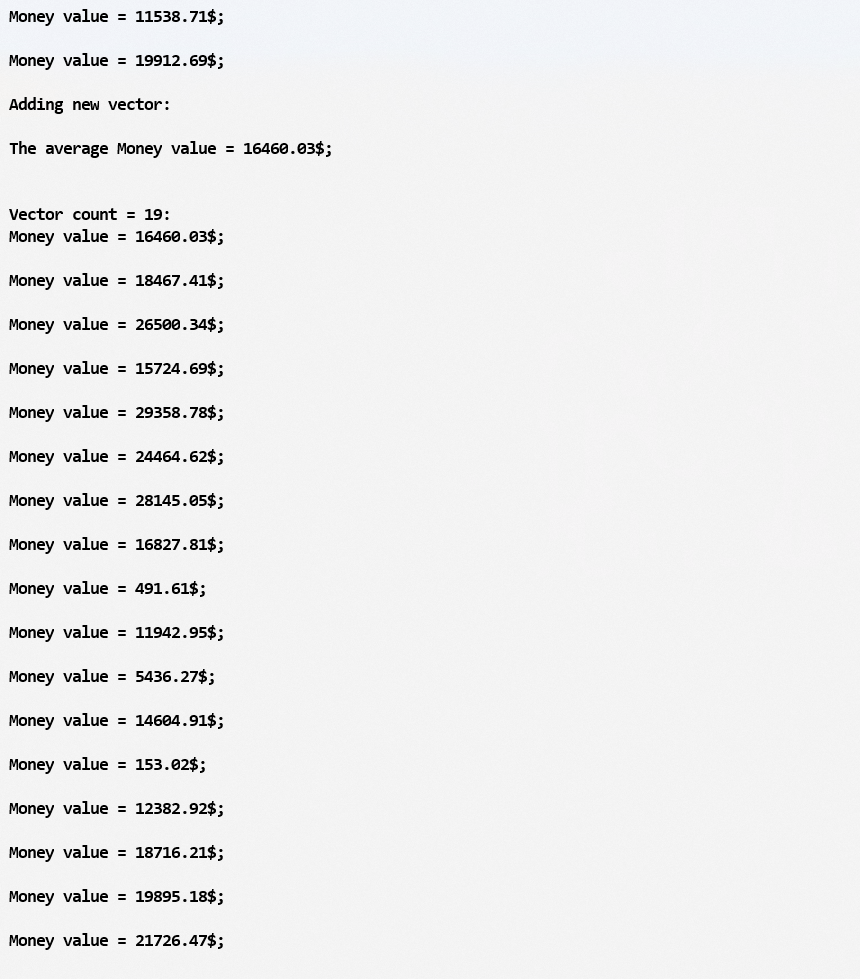
PrintVector(newVector);

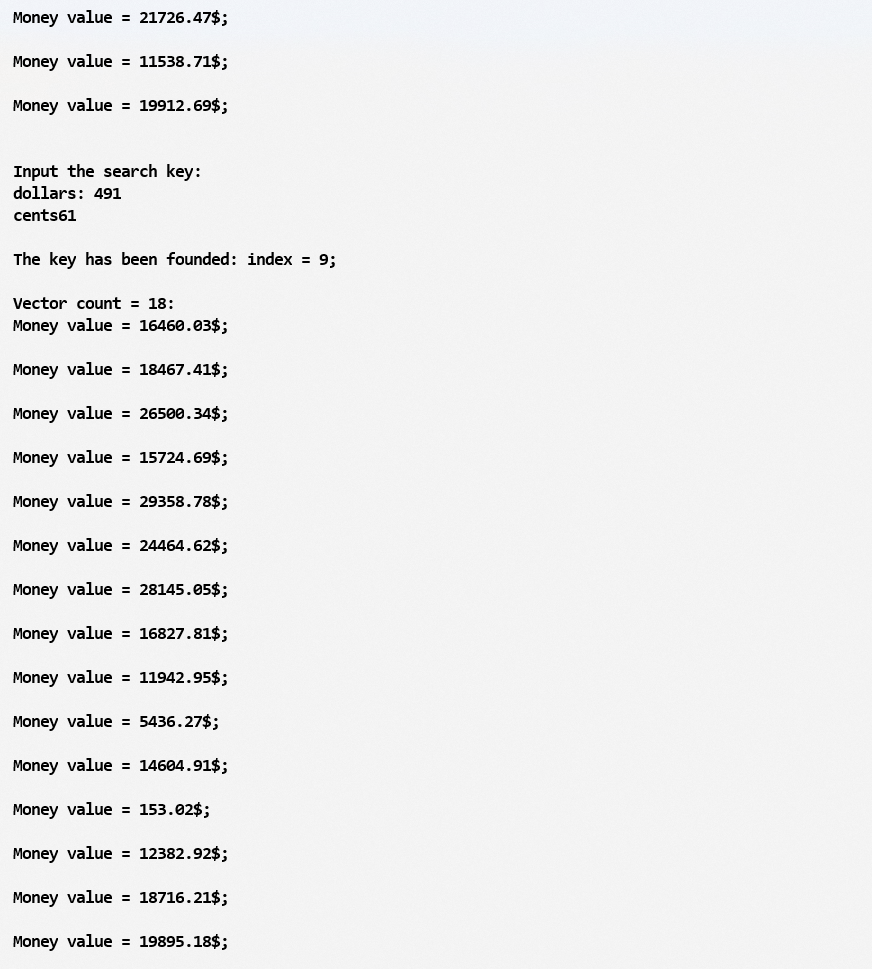
return 0;

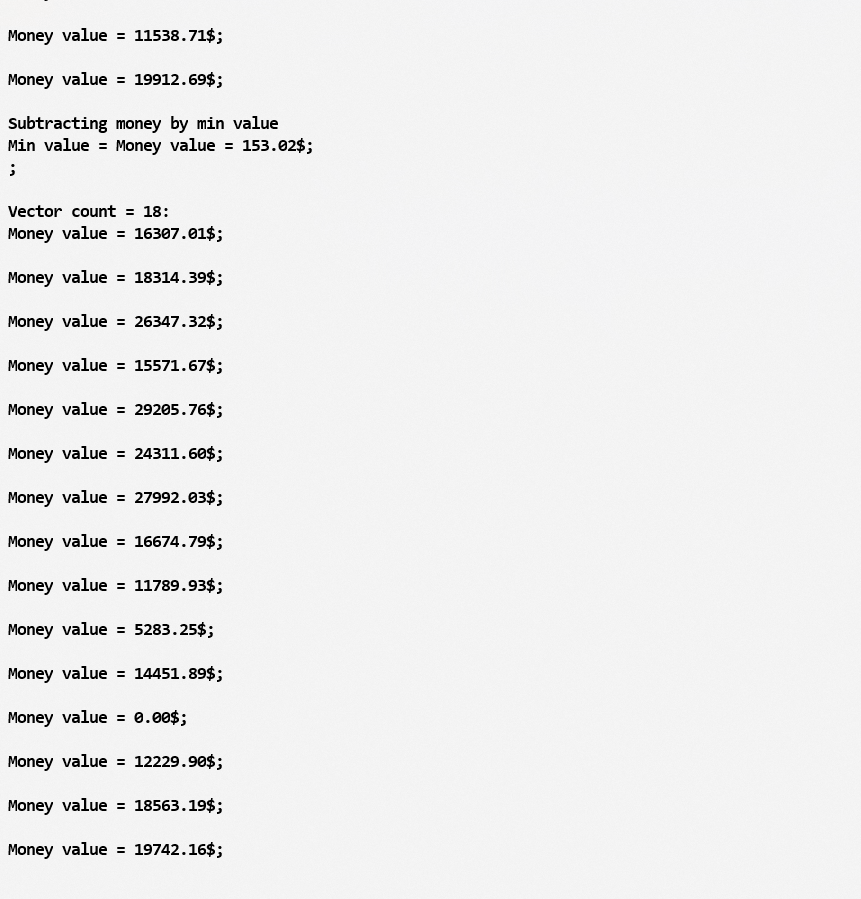
}

**Скриншоты выполненной программы:**









**Задание №3:**

К вышеупомянутому функционалу прибавляется необходимость использовать пользовательский список вместо глобальных функций в программе.

Для решения этой задачи можно воспользоваться списком, реализованным в предыдущих лабораторных работах.

Поскольку класс является параметризованным, то вся реализация находится внутри ОДНОГО заголовочного файла.

#pragma once

#include <iostream>

using namespace std;

template <class T>

class ListInt

{

private:

T\* \_data;

int \_length;

void MakeList(int length, T fillValue);

public:

ListInt();

ListInt(int length, T fillValue);

ListInt(const ListInt<T>& copied);

~ListInt();

void PrintData();

void PrintItem(int index);

void InputData();

void ReInputItem(int index);

void FindAverageAndPushFront();

void FindAndDelete(T key);

void SubtractByMinValue();

void DeleteByIndex(int index);

T& operator [] (int index);

ListInt& operator = (const ListInt<T>& assigned);

ListInt operator + (const ListInt<T>& adder);

template <class T>

friend istream& operator >> (istream& in, ListInt& list);

template <class T>

friend ostream& operator << (ostream& out, ListInt& list);

int Length() { return \_length; }

};

template<class T>

void ListInt<T>::MakeList(int length, T fillValue)

{

\_length = length;

\_data = new T[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = fillValue;

}

template<class T>

ListInt<T>::ListInt()

{

\_length = 1;

\_data = new T[\_length];

}

template<class T>

ListInt<T>::ListInt(int length, T fillValue)

{

MakeList(length, fillValue);

}

template<class T>

ListInt<T>::ListInt(const ListInt<T>& copied)

{

\_length = copied.\_length;

\_data = new T[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = copied.\_data[i];

}

template<class T>

ListInt<T>::~ListInt()

{

delete[] \_data;

\_data = nullptr;

}

template<class T>

void ListInt<T>::PrintData()

{

cout << "\nCount of items = " << \_length << ":\n";

for (int i = 0; i < \_length; i++)

PrintItem(i);

}

template<class T>

void ListInt<T>::PrintItem(int index)

{

cout << "Item #" << index + 1 << ":\n";

cout << \_data[index] << endl;

}

template<class T>

void ListInt<T>::InputData()

{

cout << "Enter the count of items: ";

int count;

cin >> count;

\_length = count;

\_data = new T[count];

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

ReInputItem(i);

}

template<class T>

void ListInt<T>::ReInputItem(int index)

{

cin >> \_data[index];

}

template<class T>

void ListInt<T>::FindAverageAndPushFront()

{

T average = \_data[0];

for (int i = 1; i < \_length; i++)

average += \_data[i];

average /= \_length;

\_length++;

T\* newData = new T[\_length];

newData[0] = average;

for (int i = 0; i < \_length - 1; i++)

newData[i + 1] = \_data[i];

delete[] \_data;

\_data = newData;

}

template<class T>

void ListInt<T>::FindAndDelete(T key)

{

for (int i = \_length - 1; i >= 0; i--)

if (\_data[i] == key)

DeleteByIndex(i - 1);

}

template<class T>

void ListInt<T>::SubtractByMinValue()

{

T minValue = \_data[0];

for (int i = 1; i < \_length; i++)

if (\_data[i] < minValue)

minValue = \_data[i];

for (int i = 0; i < \_length; i++)

\_data[i] -= minValue;

}

template<class T>

void ListInt<T>::DeleteByIndex(int index)

{

T\* newData = new T[\_length - 1];

int offset = 0;

for (int i = 0; i < \_length; i++)

{

if (i == index + 1)

offset++;

else

newData[i - offset] = \_data[i];

}

\_length--;

\_data = newData;

}

template <class T>

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

{

try

{

if (index < 0)

throw - 1;

if (index >= \_length)

throw 1;

return \_data[index];

}

catch (int code)

{

switch (code)

{

case -1:

cerr << "Error: index is out of range => less than 0;\n";

break;

case 1:

cerr << "Error: index is out of range => more than length;\n";

break;

}

return \_data[0];

}

}

template <class T>

ListInt<T>& ListInt<T>::operator=(const ListInt<T>& assigned)

{

\_length = assigned.\_length;

delete[] \_data;

\_data = new T[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = assigned.\_data[i];

return \*this;

}

template <class T>

ListInt<T> ListInt<T>::operator+(const ListInt<T>& adder)

{

try

{

if (\_length != adder.\_length)

throw - 1;

ListInt<T> temporarilyList(\_length, 0);

for (int i = 0; i < \_length; i++)

temporarilyList[i] = \_data[i] + adder.\_data[i];

return temporarilyList;

}

catch (int code)

{

if (code == -1)

cerr << "Error: list length are not the same;\n";

return \*this;

}

}

template <class T>

istream& operator>>(istream& in, ListInt<T>& list)

{

list.InputData();

return in;

}

template <class T>

ostream& operator<<(ostream& out, ListInt<T>& list)

{

list.PrintData();

return out;

}

Тогда основная программа тоже меняется, добавлены подсказки и дополнительные объяснения для пользователя:

#include "ListInt.h"

#include "Money.h"

using namespace std;

int main()

{

cout << "Input new money:\n";

ListInt<Money> justMoney;

cin >> justMoney;

cout << justMoney;

cout << "Searching average money and push front:\n";

justMoney.FindAverageAndPushFront();

cout << justMoney;

Money notSuitMoney;

cout << "\nWhich money does not suit you?\n";

cin >> notSuitMoney;

justMoney.FindAndDelete(notSuitMoney);

cout << justMoney;

cout << "Subtracting the money:\n";

justMoney.SubtractByMinValue();

cout << justMoney;

cout << "Input new float number:\n";

ListInt<double> justNumber;

cin >> justNumber;

cout << justNumber;

cout << "Searching average money and push front:\n";

justNumber.FindAverageAndPushFront();

cout << justNumber;

cout << "\nWhich number do you wanna to delete?\n";

double key;

cin >> key;

justNumber.FindAndDelete(key);

cout << justNumber;

cout << "Subtracting the nubmer:\n";

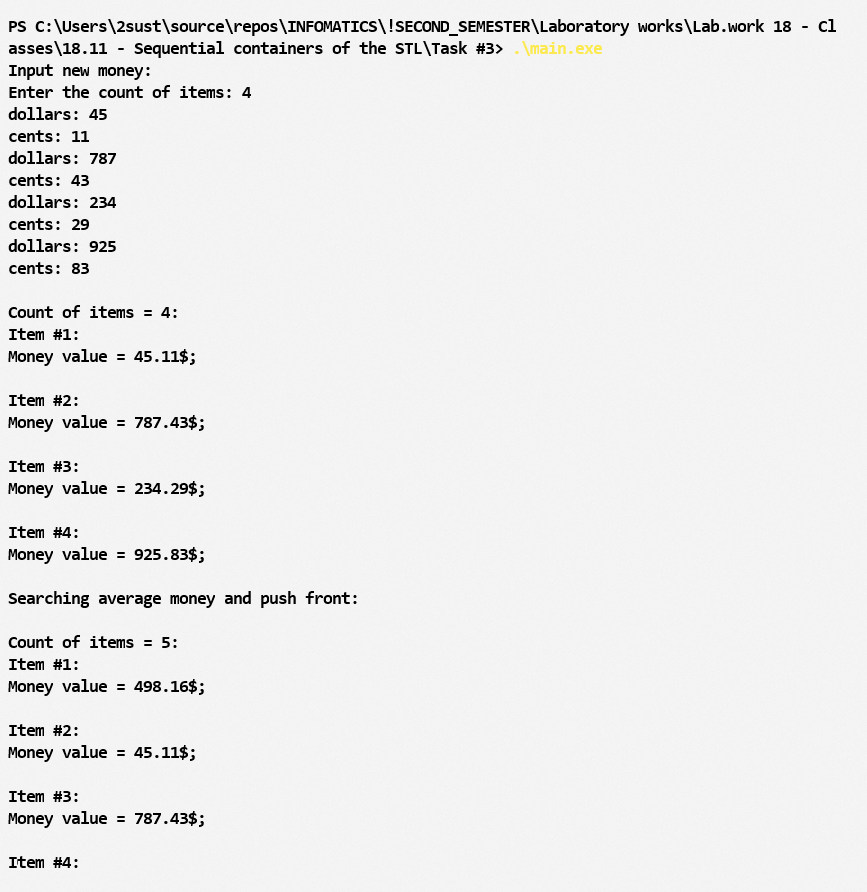
justNumber.SubtractByMinValue();

cout << justNumber;

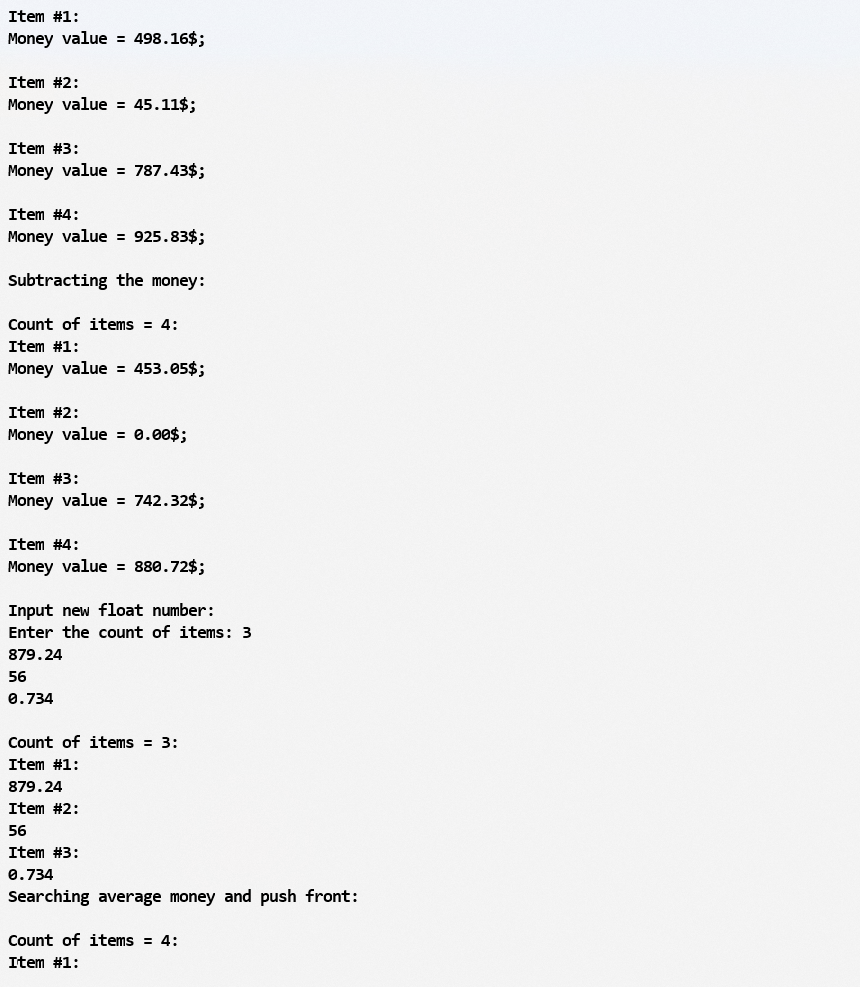
return 0;

}

**Скриншоты выполненной программы:**









**Задание №4:**

Задание идентично первому за одним исключением: вместо вектора необходимо использовать ОЧЕРЕДъ.

Соответственно, переделаем функции таким образом, чтобы они подходили для работы с очередью

Формирование случайной очереди:

void CreateRandomQueue(queue<float>& procQueue, int count)

{

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

procQueue.push(RandomFloat(0, 1000));

}

Вывод очереди на экран:

void PrintQueue(queue<float> printQueue)

{

int count = printQueue.size();

cout << "\nCount of the queue = " << count << ":\n\n";

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

{

cout << "- #" << i + 1 << " = " << printQueue.front() << ";\n";

printQueue.pop();

}

cout << endl;

}

Нахождение среднего арифметического и добавление в начало очереди:

void FindAverageAndPushFront(queue<float>& procQueue)

{

float summ = 0;

int count = procQueue.size();

queue<float> buffer;

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

{

summ += procQueue.front();

buffer.push(procQueue.front());

procQueue.pop();

}

float average = summ / count;

cout << "Average value = " << average << endl;

procQueue.push(average);

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

{

procQueue.push(buffer.front());

buffer.pop();

}

}

Нахождение и удаление элемента очереди по ключу:

void FindAndDelete(queue<float>& procQueue, float key)

{

queue<float> buffer;

int count = procQueue.size();

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

{

if (!IsFloatEqual(procQueue.front(), key))

buffer.push(procQueue.front());

else

cout << "Item found: index = " << i + 1 << ";\n";

procQueue.pop();

}

procQueue = buffer;

}

Почленное вычитание минимального элемента очереди;

void SubtractMinValue(queue<float>& procQueue)

{

queue<float> buffer = procQueue;

float minValue = buffer.front();

int count = procQueue.size();

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

{

if (buffer.front() < minValue)

minValue = buffer.front();

buffer.pop();

}

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

{

buffer.push(procQueue.front() - minValue);

procQueue.pop();

}

procQueue = buffer;

}

Основная программа идентична первой, только изменился тип данных:

int main()

{

queue<float> floatQueue;

CreateRandomQueue(floatQueue, 18);

PrintQueue(floatQueue);

FindAverageAndPushFront(floatQueue);

PrintQueue(floatQueue);

float key;

cout << "Enter the number, which you wanna to delete:";

cin >> key;

FindAndDelete(floatQueue, key);

PrintQueue(floatQueue);

SubtractMinValue(floatQueue);

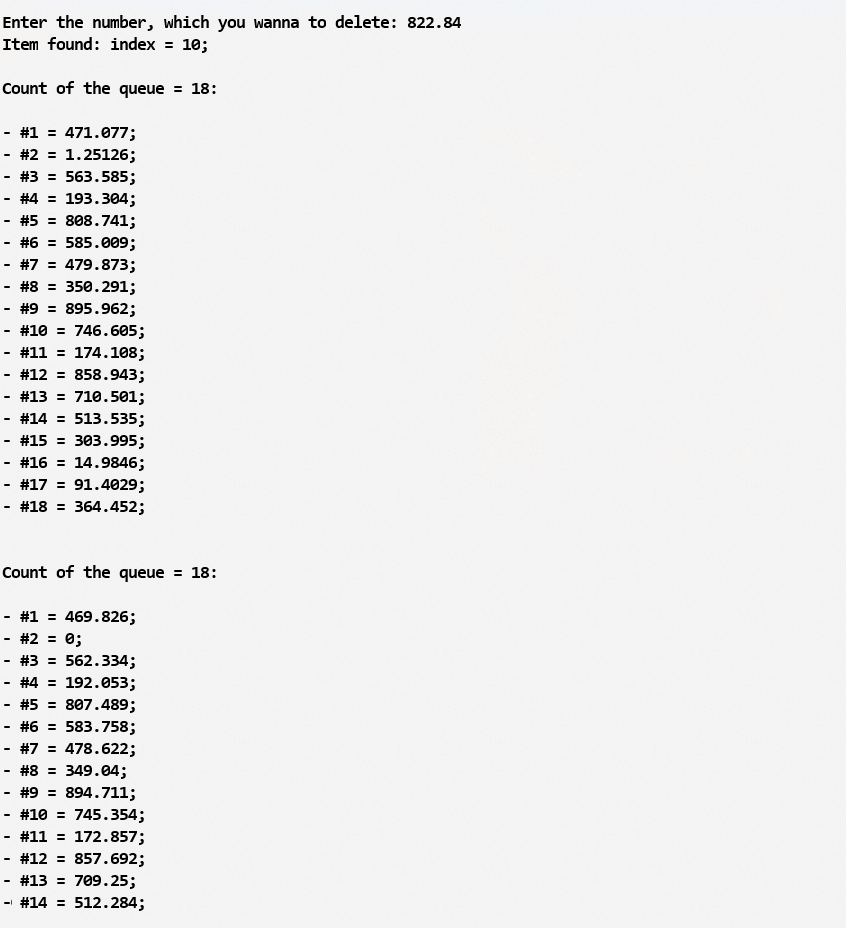
PrintQueue(floatQueue);

return 0;

}

Скриншоты выполненной программы:





Задание №5:

Задание идентично заданию №3, но вместо вектора используется очередь, класс всё такой же – параметризованный.

#pragma once

#include <iostream>

#include <queue>

using namespace std;

**//класс параметризованный**

template <class T>

class ListInt

{

private:

**//сами данные, очередь и её длина**

queue<T> \_data;

int \_length;

void MakeList(int length, T fillValue);

public:

ListInt();

ListInt(int length, T fillValue);

ListInt(const ListInt<T>& copied);

~ListInt();

void PrintData();

void InputData();

void FindAverageAndPushFront();

void FindAndDelete(T key);

void SubtractByMinValue();

ListInt& operator = (const ListInt<T>& assigned);

ListInt operator + (const ListInt<T>& adder);

template <class T>

friend istream& operator >> (istream& in, ListInt& list);

template <class T>

friend ostream& operator << (ostream& out, ListInt& list);

int Length() { return \_length; }

};

**//метод, который формирует очередь с одинаковыми элементами**

template<class T>

void ListInt<T>::MakeList(int length, T fillValue)

{

\_length = length;

for (int i = 0; i < \_length; i++)

\_data.push(fillValue);

}

**//пустой коструктор**

template<class T>

ListInt<T>::ListInt()

{

\_length = 0;

}

template<class T>

ListInt<T>::ListInt(int length, T fillValue)

{

MakeList(length, fillValue);

}

template<class T>

ListInt<T>::ListInt(const ListInt<T>& copied)

{

\_length = copied.\_length;

\_data = copied.\_data;

}

template<class T>

ListInt<T>::~ListInt()

{

}

**//Вывод очереди на экран через стоблик**

template<class T>

void ListInt<T>::PrintData()

{

cout << "\nCount of items = " << \_length << ":\n";

queue<T> copiedQueue = \_data;

for (int i = 0; i < \_length; i++)

{

cout << copiedQueue.front() << endl;

copiedQueue.pop();

}

}

**//Метод, отвечающий за ввода данных с клавиатуры**

template<class T>

void ListInt<T>::InputData()

{

cout << "Enter the count of items: ";

int count;

cin >> count;

\_length = count;

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

{

T newItem;

cin >> newItem;

\_data.push(newItem);

}

}

**//метод, который находит среднее арифметическое в очереди и добавляет его в начало очереди**

template<class T>

void ListInt<T>::FindAverageAndPushFront()

{

T summ = \_data.front();

summ -= \_data.front();

int count = \_data.size();

queue<T> buffer;

**//сумма всех элементов**

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

{

summ += \_data.front();

buffer.push(\_data.front());

\_data.pop();

}

**//само среднее арифметическое (любой тип)**

T average = summ / count;

cout << "Average value = " << average << endl;

**//добавляется среднее арифметическое в начало очереди**

\_data.push(average);

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

{

\_data.push(buffer.front());

buffer.pop();

}

\_length++;

}

**//нахождение и удаление элемента по его ключу**

template<class T>

void ListInt<T>::FindAndDelete(T key)

{

queue<T> buffer;

int count = \_data.size();

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

{

if (\_data.front() != key)

buffer.push(\_data.front());

else

cout << "Item found: index = " << i + 1 << ";\n";

\_data.pop();

}

\_data = buffer;

\_length = buffer.size();

}

**//почленное вычитание минимального элемента очереди**

template<class T>

void ListInt<T>::SubtractByMinValue()

{

queue<T> buffer = \_data;

T minValue = buffer.front();

int count = \_data.size();

**//минимальный элемента**

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

{

if (buffer.front() < minValue)

minValue = buffer.front();

buffer.pop();

}

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

{

**//почленное вычитание**

buffer.push(\_data.front() - minValue);

\_data.pop();

}

\_data = buffer;

}

template <class T>

ListInt<T>& ListInt<T>::operator=(const ListInt<T>& assigned)

{

\_length = assigned.\_length;

delete[] \_data;

\_data = new T[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = assigned.\_data[i];

return \*this;

}

template <class T>

ListInt<T> ListInt<T>::operator+(const ListInt<T>& adder)

{

try

{

if (\_length != adder.\_length)

throw - 1;

ListInt<T> temporarilyList(\_length, 0);

for (int i = 0; i < \_length; i++)

temporarilyList[i] = \_data[i] + adder.\_data[i];

return temporarilyList;

}

catch (int code)

{

if (code == -1)

cerr << "Error: list length are not the same;\n";

return \*this;

}

}

template <class T>

istream& operator>>(istream& in, ListInt<T>& list)

{

list.InputData();

return in;

}

template <class T>

ostream& operator<<(ostream& out, ListInt<T>& list)

{

list.PrintData();

return out;

}

Класс «Деньги» остаётся неизменным, основная программа принимает следующий вид:

#include "Money.h"

#include "ListInt.h"

#include <iostream>

using namespace std;

int main()

{

cout << "Input money:\n";

ListInt<Money> justMoney;

cin >> justMoney;

cout << justMoney;

cout << "Find average money and push front:\n";

justMoney.FindAverageAndPushFront();

cout << justMoney;

cout << "Searching inputted key and delete this:\n";

cout << "Enter the money, you wanna to delete:\n";

Money key;

cin >> key;

justMoney.FindAndDelete(key);

cout << justMoney;

cout << "Subtraction by minimum value\n";

justMoney.SubtractByMinValue();

cout << justMoney;

cout << "\n\n\nInput integer:\n";

ListInt<int> justInteger;

cin >> justInteger;

cout << justInteger;

cout << "Find average number and push front:\n";

justInteger.FindAverageAndPushFront();

cout << justInteger;

cout << "Searching inputted key and delete this:\n";

cout << "Enter the money, you wanna to delete:\n";

int intKey;

cin >> intKey;

justInteger.FindAndDelete(intKey);

cout << justInteger;

cout << "Subtraction by minimum value\n";

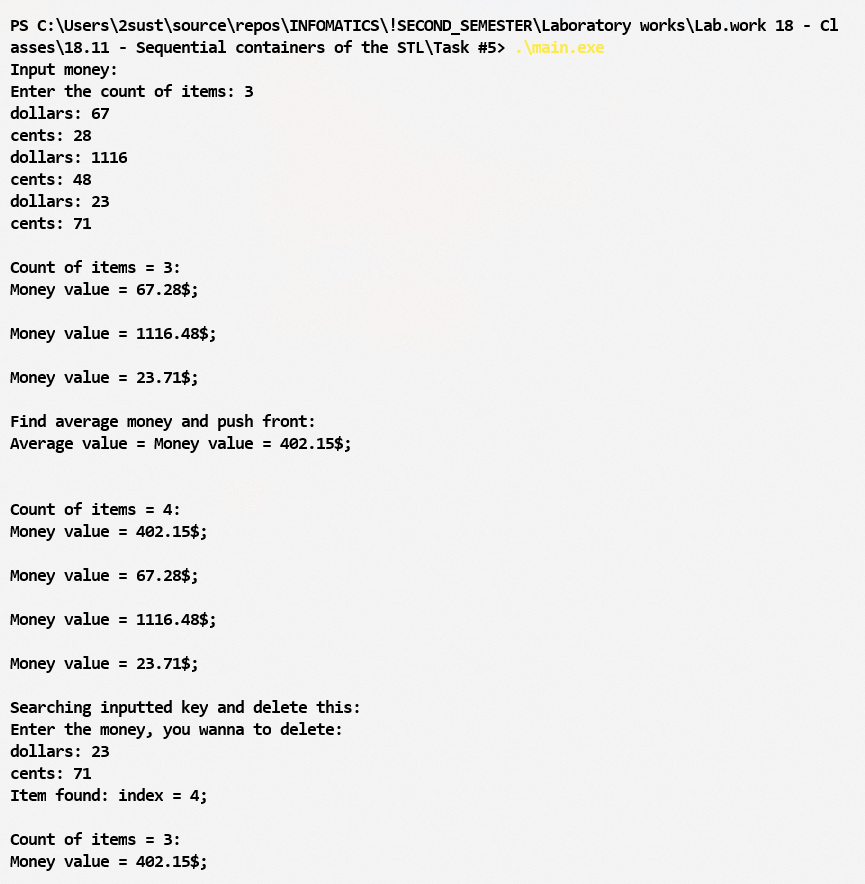
justInteger.SubtractByMinValue();

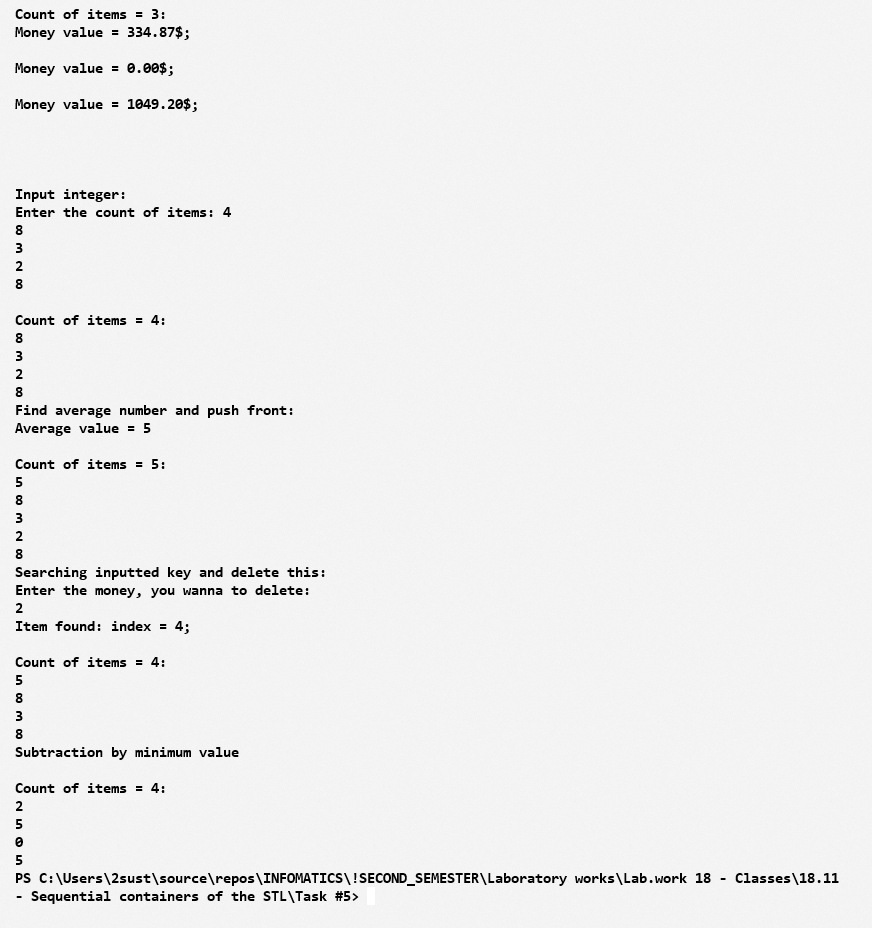
cout << justInteger;

return 0;

}

Скриншоты выполненной программы:





**Исходные коды заданий на языке программирования C++:**

Задание №1:

VectorProcessorFloat.h

#pragma once

#include <vector>

#include <iostream>

using namespace std;

vector<float>::iterator IteratorByIndex(vector<float>& processingVector, int index);

float RandomFloat(float minValue, float maxValue);

void CreateRandomVector(vector<float>& processingVector, int count);

void PrintVector(vector<float>& processingVector);

void FindAverageAndPushFront(vector<float>& processingVector);

void FindAndDelete(vector<float>& processingVector, float searchKey);

void SubtractByMinValue(vector<float>& processingVector);

VectorProcessorFloat.cpp

#include "VectorProcessorFloat.h"

#include <math.h>

vector<float>::iterator IteratorByIndex(vector<float>& processingVector, int index)

{

vector<float>::iterator iterator = processingVector.begin();

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

iterator++;

return iterator;

}

float RandomFloat(float minValue, float maxValue)

{

return minValue + static\_cast <float> (rand()) /

(static\_cast <float>(RAND\_MAX / (maxValue-minValue)));

}

void CreateRandomVector(vector<float>& processingVector, int count)

{

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

processingVector.push\_back(RandomFloat(0, 100));

}

void PrintVector(vector<float>& processingVector)

{

cout << "\nVector count = " << processingVector.size() << ":\n";

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

cout << "#" << i + 1 << ": " << processingVector[i] << ";\n";

}

void FindAverageAndPushFront(vector<float>& processingVector)

{

float summ = 0;

vector<float> temporary;

temporary.push\_back(0);

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

{

summ += processingVector[i];

temporary.push\_back(processingVector[i]);

}

float average = summ / processingVector.size();

cout << "\nThe average = " << average << ";\n";

temporary[0] = average;

processingVector = temporary;

}

void FindAndDelete(vector<float>& processingVector, float searchKey)

{

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

if (abs(processingVector[i] - searchKey) <= 0.001)

{

cout << "\nThe key has been founded: index = " << i + 1<< ";\n";

processingVector.erase(IteratorByIndex(processingVector, i));

break;

}

}

void SubtractByMinValue(vector<float>& processingVector)

{

float minValue = processingVector[0];

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

if (processingVector[i] < minValue)

minValue = processingVector[i];

cout << "\nMin value = " << minValue << ";\n";

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

processingVector[i] -= minValue;

}

main.cpp

#include "VectorProcessorFloat.h"

using namespace std;

int main()

{

vector<float> newVector;

CreateRandomVector(newVector, 18);

PrintVector(newVector);

FindAverageAndPushFront(newVector);

PrintVector(newVector);

cout << "\nInput the search key: ";

float key; cin >> key;

FindAndDelete(newVector, key);

PrintVector(newVector);

SubtractByMinValue(newVector);

PrintVector(newVector);

return 0;

}

Задание №2:

Money.h

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

class Money

{

private:

long \_dollars;

int \_cents;

public:

int Dollars() { return \_dollars; }

void SetDollars(long dollars);

int Cents() { return \_cents; }

void SetCents(int cents);

void PrintMoney();

friend bool operator > (Money& first, Money& second);

friend bool operator < (Money& first, Money& second);

friend bool operator == (Money& first, Money& second);

friend istream& operator >> (istream& in, Money& money);

friend ostream& operator << (ostream& out, Money& money);

friend fstream& operator >> (fstream& fin, Money& money);

friend fstream& operator << (fstream& fout, Money& money);

Money& operator++ ();

Money operator++ (int);

Money& operator = (const Money& value);

Money& operator += (Money& adder);

Money& operator -= (Money& subtractor);

Money& operator / (Money& divider);

Money();

Money(long dollars, int cents);

Money(const Money& parentMoney);

~Money();

};

Money.cpp

#include "Money.h"

#include <string>

using namespace std;

void Money::SetDollars(long dollars)

{

if (dollars < 0)

{

cout << "Error: dollars less than 0;\n";

\_dollars = 0;

}

else

\_dollars = dollars;

}

void Money::SetCents(int cents)

{

if (cents < 0 || cents >= 100)

{

cout << "Error: cents less than 0 or bigger than 100;\n";

\_cents = 0;

}

else

\_cents = cents;

}

void Money::PrintMoney()

{

string centStr = (\_cents < 10) ? '0' + to\_string(\_cents) : to\_string(\_cents);

cout << "Money value = " << \_dollars << '.' << centStr << "$;\n";

}

Money& Money::operator++()

{

\_cents++;

if (\_cents == 100)

{

\_cents = 0;

\_dollars++;

}

return \*this;

}

Money Money::operator++(int)

{

Money temp(\_dollars, \_cents);

++(\*this);

return temp;

}

Money::Money()

{

\_dollars = 0;

\_cents = 0;

}

Money::Money(long dollars, int cents)

{

SetDollars(dollars);

SetCents(cents);

}

Money::Money(const Money& parentMoney)

{

\_dollars = parentMoney.\_dollars;

\_cents = parentMoney.\_cents;

}

Money::~Money() {}

bool operator>(Money& first, Money& second)

{

if (first.Dollars() == second.Dollars())

return first.Cents() > second.Cents();

else

return first.Dollars() > second.Dollars();

}

bool operator<(Money& first, Money& second)

{

if (first.Dollars() == second.Dollars())

return first.Cents() < second.Cents();

else

return first.Dollars() < second.Dollars();

}

istream& operator>>(istream& in, Money& money)

{

long dollars;

int cents;

cout << "dollars: ";

cin >> dollars;

cout << "cents";

cin >> cents;

money.SetDollars(dollars);

money.SetCents(cents);

return in;

}

ostream& operator<<(ostream& out, Money& money)

{

money.PrintMoney();

return out;

}

fstream& operator>>(fstream& fin, Money& money)

{

long dollars;

int cents;

fin >> dollars >> cents;

money.SetDollars(dollars);

money.SetCents(cents);

return fin;

}

fstream& operator<<(fstream& fout, Money& money)

{

fout << money.Dollars() << endl << money.Cents() << endl;

return fout;

}

bool operator==(Money& first, Money& second)

{

return (first.Dollars() == second.Dollars()) && (first.Cents() == second.Cents());

}

Money& Money::operator+=(Money& adder)

{

int addedCents = \_cents + adder.\_cents;

if (addedCents >= 100)

{

\_dollars++;

addedCents -= 100;

}

\_dollars += adder.\_dollars;

\_cents = addedCents;

return \*this;

}

Money& Money::operator-=(Money& subtractor)

{

this->SetCents(\_cents - subtractor.Cents());

this->SetDollars(\_dollars - subtractor.Dollars());

return \*this;

}

Money& Money::operator=(const Money& value)

{

this->SetDollars(value.\_dollars);

this->SetCents(value.\_cents);

return \*this;

}

VectorProcessorMoney.h

#pragma once

#include <vector>

#include <iostream>

#include "Money.h"

using namespace std;

vector<Money>::iterator IteratorByIndex(vector<Money>& processingVector, int index);

Money RandomMoney();

void CreateRandomVector(vector<Money>& processingVector, int count);

void PrintVector(vector<Money>& processingVector);

void FindAverageAndPushFront(vector<Money>& processingVector);

void FindAndDelete(vector<Money>& processingVector, Money searchKey);

void SubtractByMinValue(vector<Money>& processingVector);

VectorProcessorMoney.cpp

#include "VectorProcessorMoney.h"

Money RandomMoney()

{

Money newMoney(rand(), rand() % 100);

return newMoney;

}

vector<Money>::iterator IteratorByIndex(vector<Money>& processingVector, int index)

{

vector<Money>::iterator iterator = processingVector.begin();

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

iterator++;

return iterator;

}

void CreateRandomVector(vector<Money>& processingVector, int count)

{

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

processingVector.push\_back(RandomMoney());

}

void PrintVector(vector<Money>& processingVector)

{

cout << "\nVector count = " << processingVector.size() << ":\n";

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

cout << processingVector[i] << "\n";

}

void FindAverageAndPushFront(vector<Money>& processingVector)

{

Money summ;

vector<Money> temporary;

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

{

summ += processingVector[i];

temporary.push\_back(processingVector[i]);

}

Money div(processingVector.size(), 0);

Money average = Money(summ.Dollars() / processingVector.size(), summ.Cents() / processingVector.size());

cout << "\nThe average " << average << "\n";

processingVector.clear();

processingVector.push\_back(average);

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

processingVector.push\_back(temporary[i]);

}

void FindAndDelete(vector<Money>& processingVector, Money searchKey)

{

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

if (processingVector[i] == searchKey)

{

cout << "\nThe key has been founded: index = " << i + 1 << ";\n";

processingVector.erase(IteratorByIndex(processingVector, i));

break;

}

}

void SubtractByMinValue(vector<Money>& processingVector)

{

Money minValue = processingVector[0];

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

if (processingVector[i] < minValue)

minValue = processingVector[i];

cout << "\nMin value = " << minValue << ";\n";

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

processingVector[i] -= minValue;

}

Задание №3:

ListInt.h

#pragma once

#include <iostream>

using namespace std;

template <class T>

class ListInt

{

private:

T\* \_data;

int \_length;

void MakeList(int length, T fillValue);

public:

ListInt();

ListInt(int length, T fillValue);

ListInt(const ListInt<T>& copied);

~ListInt();

void PrintData();

void PrintItem(int index);

void InputData();

void ReInputItem(int index);

void FindAverageAndPushFront();

void FindAndDelete(T key);

void SubtractByMinValue();

void DeleteByIndex(int index);

T& operator [] (int index);

ListInt& operator = (const ListInt<T>& assigned);

ListInt operator + (const ListInt<T>& adder);

template <class T>

friend istream& operator >> (istream& in, ListInt& list);

template <class T>

friend ostream& operator << (ostream& out, ListInt& list);

int Length() { return \_length; }

};

template<class T>

void ListInt<T>::MakeList(int length, T fillValue)

{

\_length = length;

\_data = new T[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = fillValue;

}

template<class T>

ListInt<T>::ListInt()

{

\_length = 1;

\_data = new T[\_length];

}

template<class T>

ListInt<T>::ListInt(int length, T fillValue)

{

MakeList(length, fillValue);

}

template<class T>

ListInt<T>::ListInt(const ListInt<T>& copied)

{

\_length = copied.\_length;

\_data = new T[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = copied.\_data[i];

}

template<class T>

ListInt<T>::~ListInt()

{

delete[] \_data;

\_data = nullptr;

}

template<class T>

void ListInt<T>::PrintData()

{

cout << "\nCount of items = " << \_length << ":\n";

for (int i = 0; i < \_length; i++)

PrintItem(i);

}

template<class T>

void ListInt<T>::PrintItem(int index)

{

cout << "Item #" << index + 1 << ":\n";

cout << \_data[index] << endl;

}

template<class T>

void ListInt<T>::InputData()

{

cout << "Enter the count of items: ";

int count;

cin >> count;

\_length = count;

\_data = new T[count];

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

ReInputItem(i);

}

template<class T>

void ListInt<T>::ReInputItem(int index)

{

cin >> \_data[index];

}

template<class T>

void ListInt<T>::FindAverageAndPushFront()

{

T average = \_data[0];

for (int i = 1; i < \_length; i++)

average += \_data[i];

average /= \_length;

\_length++;

T\* newData = new T[\_length];

newData[0] = average;

for (int i = 0; i < \_length - 1; i++)

newData[i + 1] = \_data[i];

delete[] \_data;

\_data = newData;

}

template<class T>

void ListInt<T>::FindAndDelete(T key)

{

for (int i = \_length - 1; i >= 0; i--)

if (\_data[i] == key)

DeleteByIndex(i - 1);

}

template<class T>

void ListInt<T>::SubtractByMinValue()

{

T minValue = \_data[0];

for (int i = 1; i < \_length; i++)

if (\_data[i] < minValue)

minValue = \_data[i];

for (int i = 0; i < \_length; i++)

\_data[i] -= minValue;

}

template<class T>

void ListInt<T>::DeleteByIndex(int index)

{

T\* newData = new T[\_length - 1];

int offset = 0;

for (int i = 0; i < \_length; i++)

{

if (i == index + 1)

offset++;

else

newData[i - offset] = \_data[i];

}

\_length--;

\_data = newData;

}

template <class T>

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

{

try

{

if (index < 0)

throw - 1;

if (index >= \_length)

throw 1;

return \_data[index];

}

catch (int code)

{

switch (code)

{

case -1:

cerr << "Error: index is out of range => less than 0;\n";

break;

case 1:

cerr << "Error: index is out of range => more than length;\n";

break;

}

return \_data[0];

}

}

template <class T>

ListInt<T>& ListInt<T>::operator=(const ListInt<T>& assigned)

{

\_length = assigned.\_length;

delete[] \_data;

\_data = new T[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = assigned.\_data[i];

return \*this;

}

template <class T>

ListInt<T> ListInt<T>::operator+(const ListInt<T>& adder)

{

try

{

if (\_length != adder.\_length)

throw - 1;

ListInt<T> temporarilyList(\_length, 0);

for (int i = 0; i < \_length; i++)

temporarilyList[i] = \_data[i] + adder.\_data[i];

return temporarilyList;

}

catch (int code)

{

if (code == -1)

cerr << "Error: list length are not the same;\n";

return \*this;

}

}

template <class T>

istream& operator>>(istream& in, ListInt<T>& list)

{

list.InputData();

return in;

}

template <class T>

ostream& operator<<(ostream& out, ListInt<T>& list)

{

list.PrintData();

return out;

}

main.cpp

#include "ListInt.h"

#include "Money.h"

using namespace std;

int main()

{

cout << "Input new money:\n";

ListInt<Money> justMoney;

cin >> justMoney;

cout << justMoney;

cout << "Searching average money and push front:\n";

justMoney.FindAverageAndPushFront();

cout << justMoney;

Money notSuitMoney;

cout << "\nWhich money does not suit you?\n";

cin >> notSuitMoney;

justMoney.FindAndDelete(notSuitMoney);

cout << justMoney;

cout << "Subtracting the money:\n";

justMoney.SubtractByMinValue();

cout << justMoney;

cout << "Input new float number:\n";

ListInt<double> justNumber;

cin >> justNumber;

cout << justNumber;

cout << "Searching average money and push front:\n";

justNumber.FindAverageAndPushFront();

cout << justNumber;

cout << "\nWhich number do you wanna to delete?\n";

double key;

cin >> key;

justNumber.FindAndDelete(key);

cout << justNumber;

cout << "Subtracting the nubmer:\n";

justNumber.SubtractByMinValue();

cout << justNumber;

return 0;

}

Задание №4:

main.cpp

#include <queue>

#include <math.h>

#include <iostream>

using namespace std;

float RandomFloat(float minValue, float maxValue)

{

return minValue + static\_cast <float> (rand()) /

(static\_cast <float>(RAND\_MAX / (maxValue-minValue)));

}

void CreateRandomQueue(queue<float>& procQueue, int count)

{

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

procQueue.push(RandomFloat(0, 1000));

}

void PrintQueue(queue<float> printQueue)

{

int count = printQueue.size();

cout << "\nCount of the queue = " << count << ":\n\n";

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

{

cout << "- #" << i + 1 << " = " << printQueue.front() << ";\n";

printQueue.pop();

}

cout << endl;

}

void FindAverageAndPushFront(queue<float>& procQueue)

{

float summ = 0;

int count = procQueue.size();

queue<float> buffer;

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

{

summ += procQueue.front();

buffer.push(procQueue.front());

procQueue.pop();

}

float average = summ / count;

cout << "Average value = " << average << endl;

procQueue.push(average);

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

{

procQueue.push(buffer.front());

buffer.pop();

}

}

bool IsFloatEqual(float first, float second)

{

return (abs(first - second) < 0.001);

}

void FindAndDelete(queue<float>& procQueue, float key)

{

queue<float> buffer;

int count = procQueue.size();

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

{

if (!IsFloatEqual(procQueue.front(), key))

buffer.push(procQueue.front());

else

cout << "Item found: index = " << i + 1 << ";\n";

procQueue.pop();

}

procQueue = buffer;

}

void SubtractMinValue(queue<float>& procQueue)

{

queue<float> buffer = procQueue;

float minValue = buffer.front();

int count = procQueue.size();

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

{

if (buffer.front() < minValue)

minValue = buffer.front();

buffer.pop();

}

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

{

buffer.push(procQueue.front() - minValue);

procQueue.pop();

}

procQueue = buffer;

}

int main()

{

queue<float> floatQueue;

CreateRandomQueue(floatQueue, 18);

PrintQueue(floatQueue);

FindAverageAndPushFront(floatQueue);

PrintQueue(floatQueue);

float key;

cout << "Enter the number, which you wanna to delete:";

cin >> key;

FindAndDelete(floatQueue, key);

PrintQueue(floatQueue);

SubtractMinValue(floatQueue);

PrintQueue(floatQueue);

return 0;

}

Задание №5:

ListInt.h

#pragma once

#include <iostream>

#include <queue>

using namespace std;

template <class T>

class ListInt

{

private:

queue<T> \_data;

int \_length;

void MakeList(int length, T fillValue);

public:

ListInt();

ListInt(int length, T fillValue);

ListInt(const ListInt<T>& copied);

~ListInt();

void PrintData();

void InputData();

void FindAverageAndPushFront();

void FindAndDelete(T key);

void SubtractByMinValue();

ListInt& operator = (const ListInt<T>& assigned);

ListInt operator + (const ListInt<T>& adder);

template <class T>

friend istream& operator >> (istream& in, ListInt& list);

template <class T>

friend ostream& operator << (ostream& out, ListInt& list);

int Length() { return \_length; }

};

template<class T>

void ListInt<T>::MakeList(int length, T fillValue)

{

\_length = length;

for (int i = 0; i < \_length; i++)

\_data.push(fillValue);

}

template<class T>

ListInt<T>::ListInt()

{

\_length = 0;

}

template<class T>

ListInt<T>::ListInt(int length, T fillValue)

{

MakeList(length, fillValue);

}

template<class T>

ListInt<T>::ListInt(const ListInt<T>& copied)

{

\_length = copied.\_length;

\_data = copied.\_data;

}

template<class T>

ListInt<T>::~ListInt()

{

}

template<class T>

void ListInt<T>::PrintData()

{

cout << "\nCount of items = " << \_length << ":\n";

queue<T> copiedQueue = \_data;

for (int i = 0; i < \_length; i++)

{

cout << copiedQueue.front() << endl;

copiedQueue.pop();

}

}

template<class T>

void ListInt<T>::InputData()

{

cout << "Enter the count of items: ";

int count;

cin >> count;

\_length = count;

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

{

T newItem;

cin >> newItem;

\_data.push(newItem);

}

}

template<class T>

void ListInt<T>::FindAverageAndPushFront()

{

T summ = \_data.front();

summ -= \_data.front();

int count = \_data.size();

queue<T> buffer;

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

{

summ += \_data.front();

buffer.push(\_data.front());

\_data.pop();

}

T average = summ / count;

cout << "Average value = " << average << endl;

\_data.push(average);

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

{

\_data.push(buffer.front());

buffer.pop();

}

\_length++;

}

template<class T>

void ListInt<T>::FindAndDelete(T key)

{

queue<T> buffer;

int count = \_data.size();

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

{

if (\_data.front() != key)

buffer.push(\_data.front());

else

cout << "Item found: index = " << i + 1 << ";\n";

\_data.pop();

}

\_data = buffer;

\_length = buffer.size();

}

template<class T>

void ListInt<T>::SubtractByMinValue()

{

queue<T> buffer = \_data;

T minValue = buffer.front();

int count = \_data.size();

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

{

if (buffer.front() < minValue)

minValue = buffer.front();

buffer.pop();

}

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

{

buffer.push(\_data.front() - minValue);

\_data.pop();

}

\_data = buffer;

}

template <class T>

ListInt<T>& ListInt<T>::operator=(const ListInt<T>& assigned)

{

\_length = assigned.\_length;

delete[] \_data;

\_data = new T[\_length];

for (int i = 0; i < \_length; i++)

\_data[i] = assigned.\_data[i];

return \*this;

}

template <class T>

ListInt<T> ListInt<T>::operator+(const ListInt<T>& adder)

{

try

{

if (\_length != adder.\_length)

throw - 1;

ListInt<T> temporarilyList(\_length, 0);

for (int i = 0; i < \_length; i++)

temporarilyList[i] = \_data[i] + adder.\_data[i];

return temporarilyList;

}

catch (int code)

{

if (code == -1)

cerr << "Error: list length are not the same;\n";

return \*this;

}

}

template <class T>

istream& operator>>(istream& in, ListInt<T>& list)

{

list.InputData();

return in;

}

template <class T>

ostream& operator<<(ostream& out, ListInt<T>& list)

{

list.PrintData();

return out;

}

main.cpp

#include "Money.h"

#include "ListInt.h"

#include <iostream>

using namespace std;

int main()

{

cout << "Input money:\n";

ListInt<Money> justMoney;

cin >> justMoney;

cout << justMoney;

cout << "Find average money and push front:\n";

justMoney.FindAverageAndPushFront();

cout << justMoney;

cout << "Searching inputted key and delete this:\n";

cout << "Enter the money, you wanna to delete:\n";

Money key;

cin >> key;

justMoney.FindAndDelete(key);

cout << justMoney;

cout << "Subtraction by minimum value\n";

justMoney.SubtractByMinValue();

cout << justMoney;

cout << "\n\n\nInput integer:\n";

ListInt<int> justInteger;

cin >> justInteger;

cout << justInteger;

cout << "Find average number and push front:\n";

justInteger.FindAverageAndPushFront();

cout << justInteger;

cout << "Searching inputted key and delete this:\n";

cout << "Enter the money, you wanna to delete:\n";

int intKey;

cin >> intKey;

justInteger.FindAndDelete(intKey);

cout << justInteger;

cout << "Subtraction by minimum value\n";

justInteger.SubtractByMinValue();

cout << justInteger;

return 0;

}

**Ответы на вопросы:**

**Из каких частей состоит библиотека STL?**

Стандартная библиотека шаблонов состоит из двух основных частей: набора контейнерных классов и набора обобщенных алгоритмов.

**Какие типы контейнеров существуют в STL?**

Контейнеры STL можно разделить на два типа: последовательные и ассоциативные.

**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;

**Что представляет собой итератор?**

Итератор - более общее понятие, чем указатель. Позволяет последовательно перемещаться по контейнеру.

**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.

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

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

**Какие типы итераторов существуют?**

В STL существуют следующие типы итераторов:

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

**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 |

**Какие операции являются эффективными для контейнера vector? Почему?**

Вставка в конец вектора, так как нет необходимости сдвигать все элементы вектора.

**Какие операции являются эффективными для контейнера list? Почему?**

Вставка и удаление на любую позицию, так как достаточно лишь изменить указатели предыдущего и следующего элементов.

**Какие операции являются эффективными для контейнера deque? Почему?**

Вставка и удаление в конец и в начало, так как, в отличие от вектора, не надо сдвигать все элементы.

**Перечислить методы, которые поддерживает последовательный контейнер vector.**

|  |  |
| --- | --- |
| 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 векторов |
| clear() | очистить  вектор |
|  |  |

**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 произвольн ое место |
| erase | удаление из произвольн  ог оместа |
|  |  |
| swap | Обмен списков |
| clear() | очистить  вектор |
| splice | сцепка  списков |

**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 |

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

Vector v(5);

For (int i=0; i<v.size(); i++) v.erase(v.begin()+i);

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

Vector v(5);

v.erase(v.end()-1);

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

List l;

For (int i=0; i<4; i++) l.pop\_back();

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

List l;

l.pop\_back();

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

deque dq;

For (int i=0; i<4; i++) dq.pop\_back();

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

deque dq;

dq.pop\_back();

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

Vector v(10);

Iterator::vector<int> I;

For (i=v.begin(); i<v.end(); i++) cout<<(\*i)

**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.

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

Первое – создает стек на базе двунаправленной очереди, второе – на базе списка.

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.

**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.

**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;

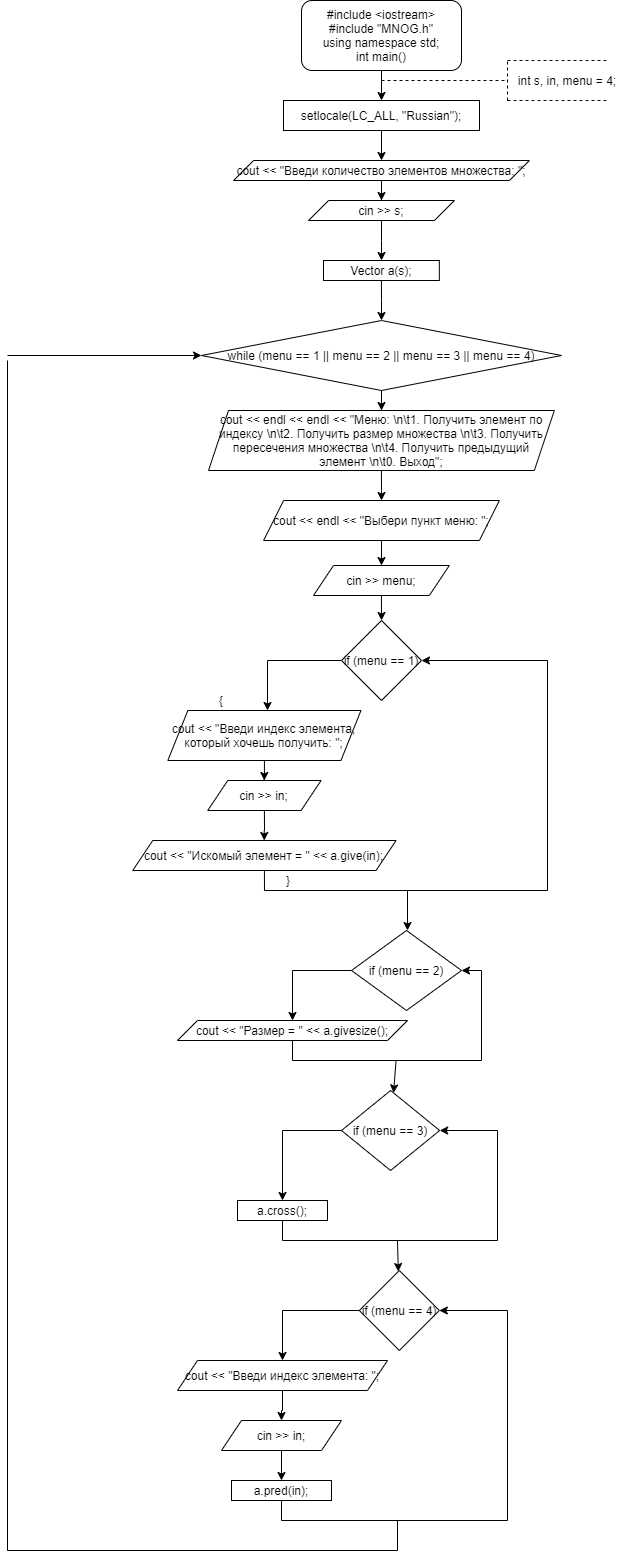
empty () – проверка пустая очередь или нет;

size() – получение размера очереди.

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

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.

Блок-схема



Vector.cpp:

