Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №18.7**

Дисциплина: «Информатика»

Тема: “Объектно-ориентированное программирование.

Шаблоны классов”

Вариант 15

Выполнил:

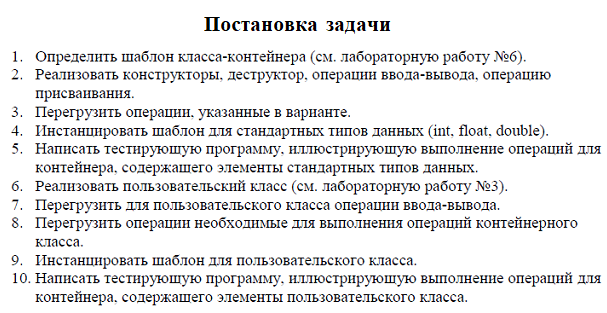
Студент группы ИВТ-20-2Б Чувашев Максим

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021



**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать шаблонный класс LIST
   2. Реализовать перегрузки операторов из варианта
2. В ходе работы были использованы следующие типы данных:
   1. Класс Pair из лабораторной работы номер 2
   2. Класс LIST

template <class T>

class LIST

{

public:

LIST()

{

Size = 0;

data = nullptr;

}

LIST(int Size) // Конструктор с параметрами

{

this->Size = Size;

data = new T[Size];

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

{

data[i] = 0;

}

}

LIST(const LIST& other) // Конструктор с параметрами

{

Size = other.Size;

if (data != nullptr)

{

delete[] data;

}

data = new T[Size];

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

{

data[i] = other.data[i];

}

}

~LIST() // деструктор

{

if (this->data != nullptr)

{

delete[] this->data;

}

data = nullptr;

}

LIST& operator = (const LIST& a)// оператор присваивания

{

if (this == &a) // Првоверка на самоприсваивание

{

return \*this;

}

Size = a.Size;

if (data != 0) // Если ячейка не пуста, то очищаем ячейку

{

delete[] data;

}

data = new int[Size];

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

{

data[i] = a.Size;

}

return \*this;

}

T& operator [] (int index) // операция доступа к индексу

{

if (index < Size)

{

return data[index];

}

else

{

cout << "\nError! Index>Size";

}

}

LIST<T>& operator \* (LIST& other)

{

LIST\* res = new LIST;

if (this->Size > other.Size)

{

int Size1 = this->Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = other.Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = this->Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0; // Заполняем разничу 1 и 2 листа нулями, так как число умножить не на что

}

return \*res;

}

else

{

int Size1 = other.Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = this->Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = other.Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0;

}

return \*res;

}

}

int operator ()()

{

return this->Size;

}

void SetSize(int size) // Сеттер размера вектора

{

this->Size = size;

}

int GetData() // Геттер данных ячейчки

{

return \*this->data;

}

int GetSize() //Геттер размера вектора

{

this->Size = size;

}

int size()

{

return Size;

}

friend ostream& operator<< <>(ostream& out, const LIST<T>& a);

friend istream& operator>> <>(istream& in, LIST<T>& a);

friend ostream& operator<< (ostream& out, const LIST<Pair>& other);

private:

int Size = 0;

T\* data;

};

1. Для решения задачи данные были представлены в следующем виде:
   1. Создается объект класса LIST типа Pair

LIST<Pair> list4(5);

1. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Аналогично лабораторной работе номер 6, для класса LIST реализованы перегрузки операторов ввода и вывода:

template <typename T>

ostream& operator<< <>(ostream& out, const LIST<T>& other)

{

for (int i = 0; i < other.Size; i++)

{

out << other.data[i] << " ";

}

return out;

}

template<typename T>

istream& operator>> <>(istream& in, LIST<T>& other)

{

for (int i = 0; i < other.Size; i++)

{

cout << "Введите " << i << " элемент списка: ";

in >> other.data[i];

}

return in;

}

ostream& operator<<(ostream& out, const LIST<Pair>& other)

{

for (int i = 0; i < other.Size; i++)

{

out << other.data[i] << " ";

}

return out;

}

1. Представленные задачи будут решены следующими действиями:
   1. Для классов Pair и LIST реализованы конструкторы и деструкторы, а также геттеры и сеттеры

Pair::Pair()

{

first = rand() % 1000;

second = rand() % 1000 / 99.9;

}

Pair::Pair(int first, double second)

{

this->first = first;

this->second = second;

}

Pair::~Pair()

{

}

int Pair::GetFirst()

{

return first;

}

double Pair::GetSecond()

{

return second;

}

void Pair::SetFirst(int first)

{

this->first = first;

}

void Pair::SetSecond(double second)

{

this->second = second;

}

LIST()

{

Size = 0;

data = nullptr;

}

LIST(int Size) // Конструктор с параметрами

{

this->Size = Size;

data = new T[Size];

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

{

data[i] = 0;

}

}

LIST(const LIST& other) // Конструктор с параметрами

{

Size = other.Size;

if (data != nullptr)

{

delete[] data;

}

data = new T[Size];

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

{

data[i] = other.data[i];

}

}

~LIST() // деструктор

{

if (this->data != nullptr)

{

delete[] this->data;

}

data = nullptr;

}

* 1. Аналогично ЛР6 реализованы требуемые перегрузки операторов для классов, используемых в решении.

LIST& operator = (const LIST& a)// оператор присваивания

{

if (this == &a) // Првоверка на самоприсваивание

{

return \*this;

}

Size = a.Size;

if (data != 0) // Если ячейка не пуста, то очищаем ячейку

{

delete[] data;

}

data = new int[Size];

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

{

data[i] = a.Size;

}

return \*this;

}

T& operator [] (int index) // операция доступа к индексу

{

if (index < Size)

{

return data[index];

}

else

{

cout << "\nError! Index>Size";

}

}

LIST<T>& operator \* (LIST& other)

{

LIST\* res = new LIST;

if (this->Size > other.Size)

{

int Size1 = this->Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = other.Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = this->Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0; // Заполняем разничу 1 и 2 листа нулями, так как число умножить не на что

}

return \*res;

}

else

{

int Size1 = other.Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = this->Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = other.Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0;

}

return \*res;

}

}

int operator ()()

{

return this->Size;

}

void SetSize(int size) // Сеттер размера вектора

{

this->Size = size;

}

int GetData() // Геттер данных ячейчки

{

return \*this->data;

}

int GetSize() //Геттер размера вектора

{

this->Size = size;

}

int size()

{

return Size;

}

friend ostream& operator<< <>(ostream& out, const LIST<T>& a);

friend istream& operator>> <>(istream& in, LIST<T>& a);

friend ostream& operator<< (ostream& out, const LIST<Pair>& other);

private:

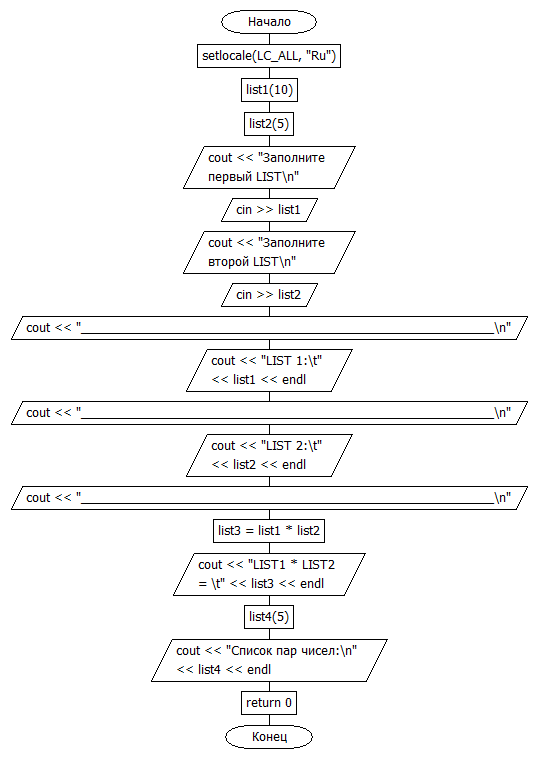
int Size = 0;

T\* data;

};

**Блок-схема**

**Source.cpp**















**Код**

#include <iostream>

#include "LIST.h"

#include "Pair.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Ru");

LIST<double> list1(10);

LIST<double> list2(5);

cout << "Заполните первый LIST\n";

cin >> list1;

cout << "Заполните второй LIST\n";

cin >> list2;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "LIST 1:\t" << list1 << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "LIST 2:\t" << list2 << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

LIST<double> list3 = list1 \* list2;

cout << "LIST1 \* LIST2 = \t" << list3 << endl;

LIST<Pair> list4(5);

cout << "Список пар чисел:\n" << list4 << endl;

return 0;

}

#include "Pair.h"

#include <iostream>

using namespace std;

Pair::Pair()

{

first = rand() % 1000;

second = rand() % 1000 / 99.9;

}

Pair::Pair(int first, double second)

{

this->first = first;

this->second = second;

}

Pair::~Pair()

{

}

int Pair::GetFirst()

{

return first;

}

double Pair::GetSecond()

{

return second;

}

void Pair::SetFirst(int first)

{

this->first = first;

}

void Pair::SetSecond(double second)

{

this->second = second;

}

bool Pair::operator!= (Pair other)

{

return this->first != other.first && this->second != other.second;

}

Pair Pair::operator=(int number)

{

Pair result;

result.first = (int)number;

result.second = number;

return result;

}

Pair Pair::operator=(double number)

{

Pair result;

result.first = (int)number;

result.second = number;

return result;

}

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

{

in >> other.first;

in >> other.second;

return in;

}

ostream& operator<<(ostream& out, Pair other)

{

out << other.first << ":" << other.second << endl;

return out;

}

#pragma once

#include <string>

#include <iostream>

#include "Pair.h"

using namespace std;

template <class T>

class LIST

{

public:

LIST()

{

Size = 0;

data = nullptr;

}

LIST(int Size) // Конструктор с параметрами

{

this->Size = Size;

data = new T[Size];

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

{

data[i] = 0;

}

}

LIST(const LIST& other) // Конструктор с параметрами

{

Size = other.Size;

if (data != nullptr)

{

delete[] data;

}

data = new T[Size];

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

{

data[i] = other.data[i];

}

}

~LIST() // деструктор

{

if (this->data != nullptr)

{

delete[] this->data;

}

data = nullptr;

}

LIST& operator = (const LIST& a)// оператор присваивания

{

if (this == &a) // Првоверка на самоприсваивание

{

return \*this;

}

Size = a.Size;

if (data != 0) // Если ячейка не пуста, то очищаем ячейку

{

delete[] data;

}

data = new int[Size];

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

{

data[i] = a.Size;

}

return \*this;

}

T& operator [] (int index) // операция доступа к индексу

{

if (index < Size)

{

return data[index];

}

else

{

cout << "\nError! Index>Size";

}

}

LIST<T>& operator \* (LIST& other)

{

LIST\* res = new LIST;

if (this->Size > other.Size)

{

int Size1 = this->Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = other.Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = this->Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0; // Заполняем разничу 1 и 2 листа нулями, так как число умножить не на что

}

return \*res;

}

else

{

int Size1 = other.Size;

res->Size = Size1;

res->data = new T[Size1];

Size1 = this->Size;

int j = 0;

for (int i = 0; i < Size1; i++)

{

res->data[i] = this->data[i] \* other.data[i];

j = i;

}

Size1 = other.Size;

j++;

for (j; j < Size1; j++)

{

res->data[j] = 0;

}

return \*res;

}

}

int operator ()()

{

return this->Size;

}

void SetSize(int size) // Сеттер размера вектора

{

this->Size = size;

}

int GetData() // Геттер данных ячейчки

{

return \*this->data;

}

int GetSize() //Геттер размера вектора

{

this->Size = size;

}

int size()

{

return Size;

}

friend ostream& operator<< <>(ostream& out, const LIST<T>& a);

friend istream& operator>> <>(istream& in, LIST<T>& a);

friend ostream& operator<< (ostream& out, const LIST<Pair>& other);

private:

int Size = 0;

T\* data;

};

template <typename T>

ostream& operator<< <>(ostream& out, const LIST<T>& other)

{

for (int i = 0; i < other.Size; i++)

{

out << other.data[i] << " ";

}

return out;

}

template<typename T>

istream& operator>> <>(istream& in, LIST<T>& other)

{

for (int i = 0; i < other.Size; i++)

{

cout << "Введите " << i << " элемент списка: ";

in >> other.data[i];

}

return in;

}

ostream& operator<<(ostream& out, const LIST<Pair>& other)

{

for (int i = 0; i < other.Size; i++)

{

out << other.data[i] << " ";

}

return out;

}

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

Pair();

Pair(int first, double second);

int GetFirst();

double GetSecond();

void SetFirst(int first);

void SetSecond(double second);

~Pair();

Pair operator+(const int other)

{

Pair result;

result.first = this->GetFirst() + this->GetSecond();

result.second = this->second;

return result;

}

Pair operator+(const double other)

{

Pair result;

result.first = this->first;

result.second = this->first + this->second;

return result;

}

Pair operator=(int rightOperand);

Pair operator=(double rightOperand);

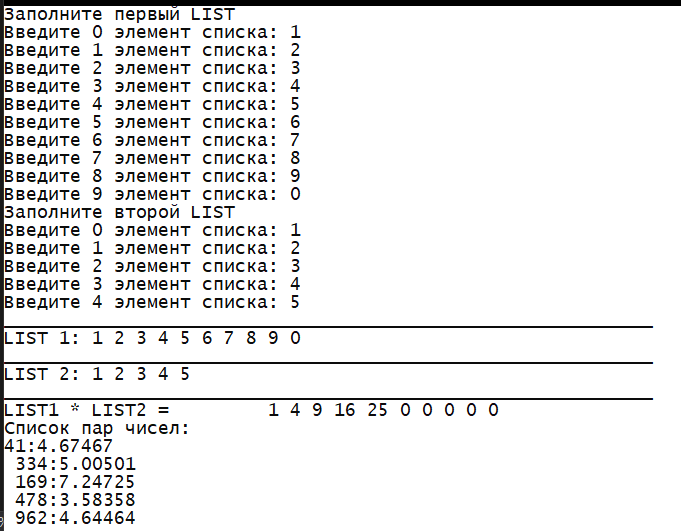
bool operator!=(Pair secondOperand);

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

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

};

**Скриншоты**

1. 

**Ответы на вопросы**

