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 №18.9**

Дисциплина: «Информатика»

Тема: “ Объектно-ориентированное программирование.

Обработка исключительных ситуаций.”

Вариант 15

Выполнил:

Студент группы ИВТ-20-2Б Чувашев Максим

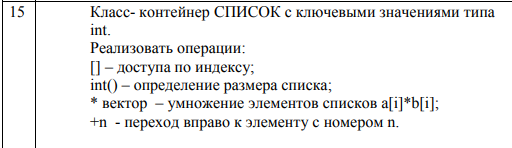
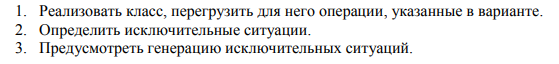
Проверила:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**



**Анализ задачи**

1. Для решения задачи необходимо:
   1. Использовать класс LIST из 7 лабораторной работы
   2. Добавить обработку исключительных ситуаций
2. В ходу работы были использованы следующие типы данных
   1. Класс LIST из 7 лабораторной работы

#pragma once

#include <string>

#include <iostream>

#include "Iterator.h"

using namespace std;

const int MAX\_SIZE = 30;//максимальный размер LIST

class LIST

{

public:

LIST();

LIST(int s); // Конструктор с параметрами

LIST(const LIST& a); // Конструктор с параметрами

~LIST(); // деструктор

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

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

LIST& operator \* (LIST& a);

LIST& operator +(int n);

int operator ()();

void SetData(int); // Сеттер данных ячейки вектора

void SetSize(int); // Сеттер размера вектора

int GetData(); // Геттер данных ячейчки

int GetSize(); //Геттер размера вектора

Iterator begin()

{

return Begin;

}

Iterator end()

{

return End;

}

int size()

{

return Size;

}

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

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

private:

friend class Iterator;

int Size = 0;

int\* data;

Iterator Current;

Iterator Begin;

Iterator End;

};

* 1. Класс Iterator:

#pragma once

#include <iostream>

#include "Error.h"

using namespace std;

class Iterator

{

friend class LIST;

public:

int\* ptr; // указатель на информационное поле элемента

Iterator() { ptr = nullptr; }// Вызов конструктора без параметров

Iterator(const Iterator& tmp) { ptr = tmp.ptr; } // Вызов конструктора копирования

Iterator& operator+(const int n) // перегрузка оператора "+"

{

if (n < 0)

{

throw IndexError1();

}

ptr += n; // сдвигаем указатель на n элементов

return \*this; // возвращаем разыменованный указатель на объект

}

void operator++() { ptr++; } // перегрузка инкримента

void operator--() { ptr--; } // перегрузка декримента

bool operator==(const Iterator& tmp) { return ptr == tmp.ptr; } // перегрузка оператора сравнения

bool operator!=(const Iterator& tmp) { return !(ptr == tmp.ptr); } // перегрузка оператора неравенства

void operator()(int\* tmp) { ptr = tmp; }// перегрузка оператора () будет возвращать размер

int& operator \*() { return \*ptr; } // перегрузка оператора разыменования

};

1. Класс Error:

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Error//базовый класс

{

public:

virtual void what() {};

};

class IndexError :public Error //ошибка в индексе вектора

{

protected:

string msg;

public:

IndexError() { msg = "Index Error\n"; }

virtual void what() { cout << msg; }

};

class SizeError :public Error //ошибка в размере вектора

{

protected:

string msg;

public:

SizeError() { msg = "size error\n"; }

virtual void what() { cout << msg; }

};

class MaxSizeError :public SizeError //превышение максимального размера

{

protected:

string msg\_;

public:

MaxSizeError() { SizeError(); msg\_ = "size>MAXSIZE\n"; }

void what() { cout << msg << msg\_; }

};

class EmptySizeError :public SizeError //удаление из пустого вектора

{

protected:

string msg\_;

public:

EmptySizeError() { SizeError(); msg\_ = "Vector is empty\n"; }

void what() { cout << msg << msg\_; }

};

class IndexError1 :public IndexError //индекс меньше нуля

{

protected:

string msg\_;

public:

IndexError1() { IndexError(); msg\_ = "index <0\n"; }

void what() { cout << msg << msg\_; }

};

class IndexError2 :public IndexError //индекс больше текущего размера вектора

{

protected:

string msg\_;

public:

IndexError2() { IndexError(); msg\_ = "index>size\n"; }

void what() { cout << msg << msg\_; }

};

1. Для решения задачи данные были представлены в следующем виде:
   1. Данные хранятся в LIST
   2. Выделены следующие исключительные ситуации:
      1. Если размер массива отрицательный бросается size error

LIST::LIST(int Size)

{

if (Size > MAX\_SIZE)

{

throw MaxSizeError();

}

else if (Size < 0)

{

throw SizeError();

}

else

{

this->Size = Size;

data = new int[Size];

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

{

data[i] = 0;

}

Begin.ptr = &(this->data[0]);

End.ptr = &(this->data[Size - 1]);

Current.ptr = Begin.ptr; //!!!!!!!!!!!!!!!!!!!!!!!!!!!!

}

}

* + 1. При попытке доступа к недоступной области памяти

//операция доступа по индексу

int& LIST::operator[](int index)

{

if(index>=Size)

{

throw IndexError2();

}

else if (index < 0)

{

throw IndexError1();

}

else

{

return data[index];

}

}

1. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Перегруженные операторы “>>” и “<<” для класса List, см. ЛР7.

**Код**

#include <iostream>

#include "Iterator.h"

#include "LIST.h"

#include "Error.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Ru");

try

{

int size = 0;

cout << "Введите размер первого LIST: ";

cin >> size;

LIST list1(size);

cout << "Введите размер второго LIST: ";

cin >> size;

LIST list2(size);

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 list3 = list1 \* list2;

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

Iterator tmp;

tmp(&list1[0]);

tmp = tmp + 2;

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

cout << \*tmp << endl;

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

}

catch (Error &ex)

{

ex.what();

}

return 0;

}

#include "LIST.h"

#include "Error.h"

LIST::LIST()

{

Size = 0;

data = nullptr;

Begin.ptr = nullptr;

End.ptr = nullptr;

Current.ptr = nullptr;

}

LIST::LIST(int Size)

{

if (Size > MAX\_SIZE)

{

throw MaxSizeError();

}

else if (Size < 0)

{

throw SizeError();

}

else

{

this->Size = Size;

data = new int[Size];

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

{

data[i] = 0;

}

Begin.ptr = &(this->data[0]);

End.ptr = &(this->data[Size - 1]);

Current.ptr = Begin.ptr; //!!!!!!!!!!!!!!!!!!!!!!!!!!!!

}

}

// Конструктор копирования

LIST::LIST(const LIST& other)

{

Size = other.Size;

if (data != nullptr)

{

delete[] data;

}

data = new int[Size];

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

{

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

}

this->Begin = other.Begin;

this->Current = other.Current;

this->End = other.End;

}

LIST::~LIST()

{

if (this->data != nullptr)

{

delete[] this->data;

}

data = nullptr;

Current.ptr = nullptr;

Begin.ptr = nullptr;

Current.ptr = nullptr;

End.ptr = nullptr;

}

int LIST::GetSize()

{

return this->Size;

}

int LIST::GetData()

{

return \*this->data;

}

void LIST::SetSize(int size)

{

this->Size = size;

}

// Операция присваивания

LIST& LIST::operator=(const LIST& a)

{

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;

}

//операция доступа по индексу

int& LIST::operator[](int index)

{

if(index>=Size)

{

throw IndexError2();

}

else if (index < 0)

{

throw IndexError1();

}

else

{

return data[index];

}

}

//перегрузка оператора "\*"

LIST& LIST::operator \*(LIST& other)

{

LIST\* res = new LIST;

if (this->Size > other.Size)

{

int Size1 = this->Size;

res->Size = Size1;

res->data = new int[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 int[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;

}

}

// определение размера списка

int LIST::operator ()()

{

return this->Size;

}

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

{

if (other.Size == 0)

{

throw EmptySizeError();

}

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

{

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

}

return out;

}

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

{

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

{

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

in >> other.data[i];

}

return in;

}

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Error//базовый класс

{

public:

virtual void what() {};

};

class IndexError :public Error //ошибка в индексе вектора

{

protected:

string msg;

public:

IndexError() { msg = "Index Error\n"; }

virtual void what() { cout << msg; }

};

class SizeError :public Error //ошибка в размере вектора

{

protected:

string msg;

public:

SizeError() { msg = "size error\n"; }

virtual void what() { cout << msg; }

};

class MaxSizeError :public SizeError //превышение максимального размера

{

protected:

string msg\_;

public:

MaxSizeError() { SizeError(); msg\_ = "size>MAXSIZE\n"; }

void what() { cout << msg << msg\_; }

};

class EmptySizeError :public SizeError //удаление из пустого вектора

{

protected:

string msg\_;

public:

EmptySizeError() { SizeError(); msg\_ = "Vector is empty\n"; }

void what() { cout << msg << msg\_; }

};

class IndexError1 :public IndexError //индекс меньше нуля

{

protected:

string msg\_;

public:

IndexError1() { IndexError(); msg\_ = "index <0\n"; }

void what() { cout << msg << msg\_; }

};

class IndexError2 :public IndexError //индекс больше текущего размера вектора

{

protected:

string msg\_;

public:

IndexError2() { IndexError(); msg\_ = "index>size\n"; }

void what() { cout << msg << msg\_; }

};

#pragma once

#include <iostream>

#include "Error.h"

using namespace std;

class Iterator

{

friend class LIST;

public:

int\* ptr; // указатель на информационное поле элемента

Iterator() { ptr = nullptr; }// Вызов конструктора без параметров

Iterator(const Iterator& tmp) { ptr = tmp.ptr; } // Вызов конструктора копирования

Iterator& operator+(const int n) // перегрузка оператора "+"

{

if (n < 0)

{

throw IndexError1();

}

ptr += n; // сдвигаем указатель на n элементов

return \*this; // возвращаем разыменованный указатель на объект

}

void operator++() { ptr++; } // перегрузка инкримента

void operator--() { ptr--; } // перегрузка декримента

bool operator==(const Iterator& tmp) { return ptr == tmp.ptr; } // перегрузка оператора сравнения

bool operator!=(const Iterator& tmp) { return !(ptr == tmp.ptr); } // перегрузка оператора неравенства

void operator()(int\* tmp) { ptr = tmp; }// перегрузка оператора () будет возвращать размер

int& operator \*() { return \*ptr; } // перегрузка оператора разыменования

};

#pragma once

#include <string>

#include <iostream>

#include "Iterator.h"

using namespace std;

const int MAX\_SIZE = 20;

class LIST

{

public:

LIST();

LIST(int s); // Конструктор с параметрами

LIST(const LIST& a); // Конструктор с параметрами

~LIST(); // деструктор

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

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

LIST& operator \* (LIST& a);

LIST& operator +(int n);

int operator ()();

void SetData(int); // Сеттер данных ячейки вектора

void SetSize(int); // Сеттер размера вектора

int GetData(); // Геттер данных ячейчки

int GetSize(); //Геттер размера вектора

Iterator begin()

{

return Begin;

}

Iterator end()

{

return End;

}

int size()

{

return Size;

}

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

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

private:

friend class Iterator;

int Size = 0;

int\* data;

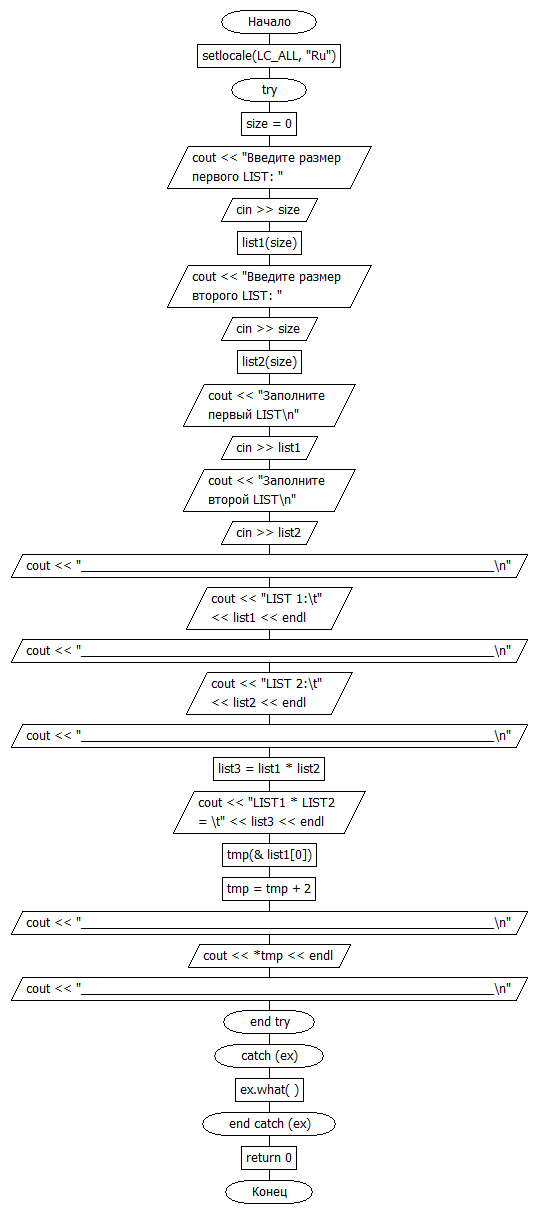
Iterator Current;

Iterator Begin;

Iterator End;

};

**Блок-схема**



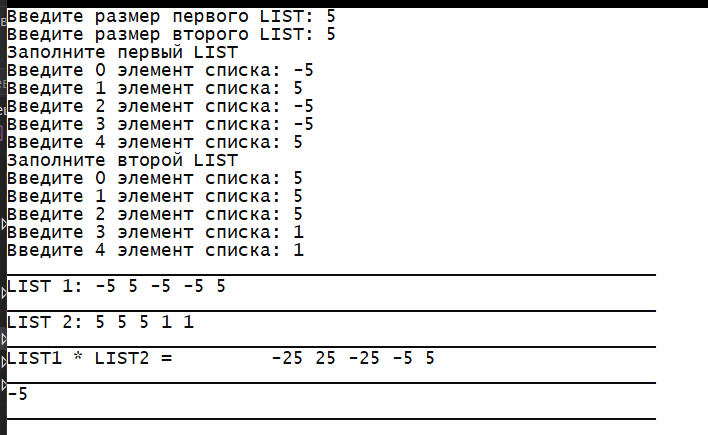








**Скриншоты**

1. 
2. 

**Ответы на вопросы**

