## **Лабораторная работа №** **1** " **Классы и объекты. Использование конструкторов. Деструкторы**"

1. Постановка задачи

Написать программу, в которой создаются и удаляются объекты определенного пользователем класса. Выполнить исследование вызовов конструкторов и деструкторов.

Класс: ИЗДЕЛИЕ

Поля:

* имя – char\*
* шифр – char\*
* количество – int

1. Код программы

Класс product

product.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class product

{

private:

string name;

string shifr;

int count;

public:

product(string name);

product(string name, string shifr, int count);

~product();

void print();

};

product.cpp

#include "product.h"

product::product(string name)

{

this->name = name;

this->shifr = "";

this->count = 0;

cout << "Вызов конструктора " << this << endl;

}

product::product(string name, string shifr, int count)

{

this->name = name;

this->shifr = shifr;

this->count = count;

cout << "Вызов конструктора " << this << endl;

}

product::~product()

{

cout << "Вызов деструктора " << this << endl;

}

void product::print()

{

cout << "Название: " << name << "; Шифр: " << shifr << "; Кол-во: " << count << endl;

}

Main.cpp

#include <iostream>

#include "product.h"

int main()

{

setlocale(LC\_ALL, "rus");

product prod1("Изделие 1");

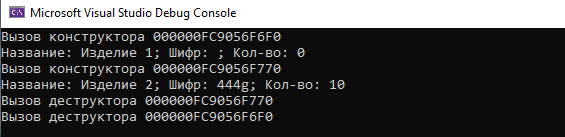
prod1.print();

product prod2("Изделие 2", "444g", 10);

prod2.print();

}

1. Результат выполнения



## **Лабораторная работа № 2**

## **Дружественные функции и классы. Перегрузка операций**

1. Постановка задачи

1. Определить пользовательский класс.

2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5. Перегрузить операцию присваивания.

6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Перегрузить операции указанные в варианте.

8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

Создать класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты отделяются от секунд двоеточием. Реализовать:

* добавление секунд (учесть, что в минуте не может быть более 60 секунд)
* сравнение временных интервалов (== и !=)

1. Код программы

Time.h

#pragma once

#include <iostream>

using namespace std;

class Time

{

private:

int min, sec;

public:

Time(); //конструктор без параметров

Time(int min, int sec); //конструктор с параметрами

Time(const Time& time); //конструктор копирования

~Time(){} //деструктор

//перегрузки операторов

//оператор присваивания

Time& operator=(const Time& t);

//оператор сложения

Time operator+(int sec);

//операторы равенства

bool operator==(const Time& t);

bool operator!=(const Time& t);

int get\_min() { return min; }

void set\_min(int min) { this->min = min; }

int get\_sec() { return sec; }

void set\_sec(int sec) { this->sec = sec; }

//перегрузка оператора ввода (cin >> t)

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

//перегрузка оператора вывода (cout << t)

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

};

Time.cpp

#include "Time.h"

Time::Time()

{

this->min = 0;

this->sec = 0;

}

Time::Time(int min, int sec)

{

this->min = min;

this->sec = sec;

}

Time::Time(const Time& time)

{

this->min = time.min;

this->sec = time.sec;

}

Time& Time::operator=(const Time& t)

{

this->min = t.min;

this->sec = t.sec;

return \*this;

}

Time Time::operator+(int sec)

{

Time res;

res.sec = this->sec + sec;

res.min = this->min;

if (res.sec >= 60)

{

res.min += res.sec / 60;

res.sec = res.sec % 60;

}

return res;

}

bool Time::operator==(const Time& t)

{

return sec == t.sec && min == t.min;

}

bool Time::operator!=(const Time& t)

{

return !(\*this == t);

}

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

{

in >> p.min >> p.sec;

return in;

}

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

{

out << p.min << ":" << p.sec;

return out;

}

Main.cpp

#include <iostream>

#include "Time.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

int min1, sec1;

cout << "Введите время (минуты и секунды): ";

cin >> min1 >> sec1;

Time time1(min1, sec1);

Time time2;

cout << "Введите время (минуты и секунды): ";

cin >> time2;

cout << "Первое время: " << time1 << endl;

cout << "Второе время: " << time2 << endl;

bool equal = time1 == time2;

if (equal)

cout << "Время одинаковое" << endl;

else

cout << "Время не одинаковое" << endl;

cout << "Сколько секунд добавить к первому времени: ";

int sec\_add;

cin >> sec\_add;

Time time3 = time1 + sec\_add;

cout << "Третье время: " << time3 << endl;

bool notequal = time2 != time3;

cout << "Результат сравнения второго и третьего времени" << endl;

if (notequal)

cout << "Время не одинаковое" << endl;

else

cout << "Время одинаковое" << endl;

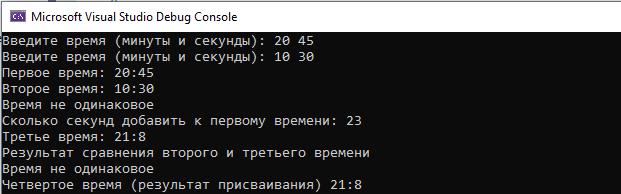
Time time4 = time3;

cout << "Четвертое время (результат присваивания) ";

cout << time4;

}

1. Результат выполнения



## **Лабораторная работа №3**

## **Наследование. Принцип подстановки**

1. Постановка задачи

1. Определить пользовательский класс.

2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5. Перегрузить операцию присваивания.

6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Определить производный класс.

8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

9. Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

Базовый класс:

ПАРА\_ЧИСЕЛ (PAIR)

Первое\_число (first) – int

Второе\_число (second) – int

Определить методы изменения полей и вычисления произведения чисел.

Создать производный класс ЭЛЛИПС (ELLIPSE), с полями: малой и большой осями. Определить метод вычисления площади.

1. Код программы

Pair.h

#pragma once

#include <iostream>

using namespace std;

class Pair

{

protected:

int first, second;

public:

Pair();

Pair(int f, int s);

Pair(const Pair& p);

~Pair(){}

Pair& operator=(const Pair& p);

int product() { return first \* second; }

int get\_first() { return first; }

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

int get\_second() { return second; }

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

//перегрузка оператора ввода (cin >> t)

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

//перегрузка оператора вывода (cout << t)

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

virtual void print() { cout << "Пара чисел: " << first << " " << second << endl; }

};

Pair.cpp

#include "Pair.h"

Pair::Pair()

{

second = 0;

first = 0;

}

Pair::Pair(int f, int s)

{

first = f;

second = s;

}

Pair::Pair(const Pair& p)

{

first = p.first;

second = p.second;

}

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

{

first = p.first;

second = p.second;

return \*this;

}

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

{

in >> p.first >> p.second;

return in;

}

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

{

out << p.first << " " << p.second;

return out;

}

Ellipse.h

#pragma once

#include "Pair.h"

class Ellipse :

public Pair

{

public:

Ellipse() : Pair(5, 10){}

Ellipse(int a, int b) : Pair(a,b){}

double get\_square();

void print() { cout << "Эллипс: a = " << first << " b = " << second << endl; }

};

Ellipse.cpp

#include "Ellipse.h"

double Ellipse::get\_square()

{

return 3.14 \* Pair::product();

}

Main.cpp

#include <iostream>

#include "Pair.h"

#include "Ellipse.h"

void change(Pair\* p)

{

p->set\_first(p->get\_first() + 1);

p->set\_second(p->get\_second() + 1);

}

Pair\* create(int k)

{

Pair\* p = nullptr;

int first = rand() % 50 + 5;

int second = rand() % 50 + 5;

if (k == 1)

p = new Pair(first, second);

else

p = new Ellipse(first, second);

return p;

}

int main()

{

setlocale(LC\_ALL, "rus");

Pair p1;

cout << "Введите два числа: ";

cin >> p1;

Ellipse el1;

cout << "Введите длины большой и малой полуоси: ";

cin >> el1;

cout << "Пара чисел: " << p1 << endl;

cout << "Эллипс: " << el1 << endl;

change(&p1);

change(&el1);

cout << "Пара чисел: " << p1 << endl;

cout << "Эллипс: " << el1 << endl;

Pair\* p2 = create(1);

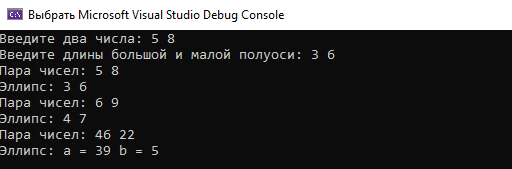
p2->print();

Pair\* el2 = create(2);

el2->print();

}

1. Результат выполнения



## **Лабораторная работа № 4**

## **Шаблоны классов**

1. Постановка задачи
   1. Определить шаблон класса-контейнера.
   2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
   3. Перегрузить операции, указанные в варианте.
   4. Инстанцировать шаблон для стандартных типов данных (int, float, double).
   5. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы стандартных типов данных.
   6. Реализовать пользовательский класс.
   7. Перегрузить для пользовательского класса операции ввода-вывода.
   8. Перегрузить операции необходимые для выполнения операций контейнерного класса.
   9. Инстанцировать шаблон для пользовательского класса.
   10. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского класса.

Класс-контейнер ВЕКТОР с элементами типа int.

Реализовать операции:

[] – доступа по индексу;

() – определение размера вектора;

\* число – умножает все элементы вектора на число;

Пользовательский класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. При выводе минуты отделяются от секунд двоеточием.

1. Код программы

Time.h

#pragma once

#include <iostream>

using namespace std;

class Time

{

private:

int min, sec;

public:

Time(); //конструктор без параметров

Time(int min, int sec); //конструктор с параметрами

Time(const Time& time); //конструктор копирования

~Time() {} //деструктор

//перегрузки операторов

//оператор присваивания

Time& operator=(const Time& t);

//оператор сложения

Time operator+(int sec);

//операторы равенства

bool operator==(const Time& t);

bool operator!=(const Time& t);

int get\_min() { return min; } //селектор

void set\_min(int min) { this->min = min; } //модификатор

int get\_sec() { return sec; } //селектор

void set\_sec(int sec) { this->sec = sec; } //модификатор

//перегрузка оператора ввода (cin >> t)

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

//перегрузка оператора вывода (cout << t)

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

Time operator\*(int k);

};

Time.cpp

#include "Time.h"

#include "Time.h"

Time::Time()

{

this->min = 0;

this->sec = 0;

}

Time::Time(int min, int sec)

{

this->min = min;

this->sec = sec;

}

Time::Time(const Time& time)

{

this->min = time.min;

this->sec = time.sec;

}

Time& Time::operator=(const Time& t)

{

this->min = t.min;

this->sec = t.sec;

return \*this;

}

Time Time::operator+(int sec)

{

Time res;

res.sec = this->sec + sec;

res.min = this->min;

if (res.sec >= 60)

{

res.min += res.sec / 60;

res.sec = res.sec % 60;

}

return res;

}

bool Time::operator==(const Time& t)

{

return sec == t.sec && min == t.min;

}

bool Time::operator!=(const Time& t)

{

return !(\*this == t);

}

Time Time::operator\*(int k)

{

int total\_sec = min \* 60 + sec;

total\_sec \*= k;

Time res;

res.min = total\_sec / 60;

res.sec = total\_sec % 60;

return res;

}

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

{

in >> p.min >> p.sec;

return in;

}

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

{

out << p.min << ":" << p.sec;

return out;

}

Vector.h

#pragma once

#include <iostream>

using namespace std;

template<class T>

class Vector

{

private:

T\* a; //массив

int size; //размер (сколько эл-тов добавлено)

int max\_size; //максим размер (сколько можно добавить элементов)

public:

Vector();

Vector(int max\_size);

Vector(const Vector<T>& v);

~Vector();

int get\_size() { return size; }

void push\_back(T data);

T& operator[](int index);

int operator()();

Vector<T> operator\*(int k);

Vector<T>& operator=(const Vector<T>& v);

//перегрузка оператора ввода (cin >> t)

friend istream& operator>> <>(istream& in, Vector<T>& v);

//перегрузка оператора вывода (cout << t)

friend ostream& operator<< <>(ostream& out, const Vector<T>& v);

};

template<class T>

Vector<T>::Vector()

{

size = 0;

max\_size = 10;

a = new T[max\_size];

}

template<class T>

Vector<T>::Vector(int max\_size)

{

size = 0;

this->max\_size = max\_size;

a = new T[max\_size];

}

template<class T>

Vector<T>::Vector(const Vector<T>& v)

{

size = v.size;

max\_size = v.max\_size;

a = new T[max\_size];

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

{

a[i] = v.a[i];

}

}

template<class T>

Vector<T>::~Vector()

{

delete[] a;

}

template<class T>

void Vector<T>::push\_back(T data)

{

if (size == max\_size)

{

//увеличиваем массив в 2 раза

T\* b = new T[size \* 2];

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

b[i] = a[i];

delete[] a;

max\_size = max\_size \* 2;

a = b;

}

a[size] = data;

size++;

}

template<class T>

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

{

return a[index];

}

template<class T>

int Vector<T>::operator()()

{

return size;

}

template<class T>

Vector<T> Vector<T>::operator\*(int k)

{

Vector<T> res(\*this);

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

{

res.a[i] = res.a[i] \* k;

}

return res;

}

template<class T>

Vector<T>& Vector<T>::operator=(const Vector<T>& v)

{

if (this != &v)

{

delete[] a;

a = new T[v.max\_size];

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

a[i] = v.a[i];

size = v.size;

max\_size = v.max\_size;

}

return \*this;

}

template<class T>

istream& operator>><>(istream& in, Vector<T>& v)

{

T r;

for (int i = 0; i < v.max\_size; i++)

{

in >> r;

v.push\_back(r);

}

return in;

}

template<class T>

ostream& operator<<<>(ostream& out, const Vector<T>& v)

{

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

{

out << v.a[i] << " ";

}

return out;

}

Main.cpp

#include <iostream>

#include "Vector.h"

#include "Time.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

Vector<int> v;

cin >> v;

cout << v << endl;

int k;

cout << "k = ";

cin >> k;

Vector<int> b = v \* k;

for (int i = 0; i < v.get\_size(); i++)

{

cout << b[i] << " ";

}

cout << endl;

cout << "Размер вектора = " << v() << endl;

Vector<Time> times(4);

cin >> times;

cout << times << endl;

cout << "k = ";

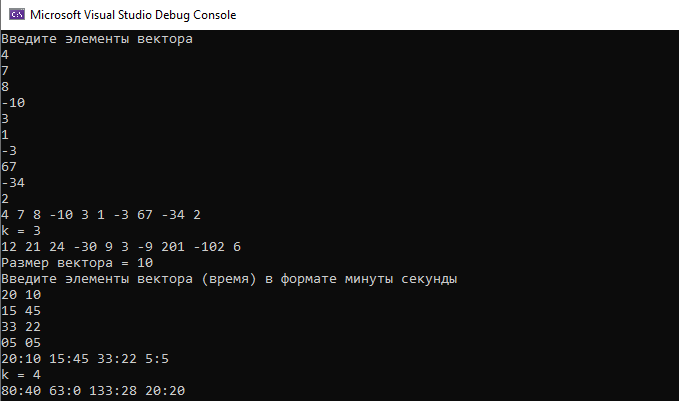
cin >> k;

Vector<Time> b\_times = times \* k;

cout << b\_times;

}

1. Результат выполнения



## **Лабораторная работа № 5**

## **Шаблоны функций**

1. Постановка задачи

Написать шаблон функций для своего варианта. Написать демонстрационную программу для вызова этих функций.

Минимальный элемент в массиве

1. Код программы

Main.cpp

#include <iostream>

#include <ctime>

using namespace std;

template<class T>

T find\_min(T\* a, int n)

{

T min = a[0];

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

{

if (a[i] < min)

min = a[i];

}

return min;

}

template<class T>

void fill\_array(T\* a, int n)

{

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

{

a[i] = rand() % 100;

}

}

template<class T>

void print\_array(T\* a, int n)

{

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

{

cout << a[i] << " ";

}

cout << endl;

}

void fill\_char\_array(char\* a, int n)

{

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

{

a[i] = (char)(rand() % 26 + 'a');

}

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(0));

int n;

cout << "введите размеры массива: ";

cin >> n;

int\* a = new int[n];

fill\_array(a, n);

print\_array(a, n);

int min1 = find\_min(a, n);

cout << "Минимальный элемент = " << min1 << endl;

double\* b = new double[n];

fill\_array(b, n);

print\_array(b, n);

double min2 = find\_min(b, n);

cout << "Минимальный элемент = " << min2 << endl;

char\* c = new char[n];

fill\_char\_array(c, n);

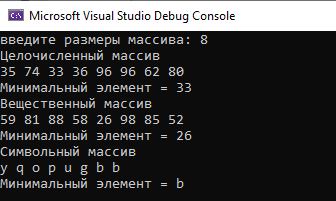
print\_array(c, n);

char min3 = find\_min(c, n);

cout << "Минимальный элемент = " << min3 << endl;

}

1. Результат выполнения



## **Лабораторная работа № 6**

## **Обработка исключительных ситуаций**

1. Постановка задачи
2. Реализовать класс, перегрузить для него операции, указанные в варианте.
3. Определить исключительные ситуации.
4. Предусмотреть генерацию исключительных ситуаций.

Класс-контейнер ВЕКТОР с элементами типа int.

Реализовать операции:

[] – доступа по индексу;

() – определение размера вектора;

\* число – умножает все элементы вектора на число;

-n – переход влево к элементу с номером n (с помощью класса-итератора).

1. Код программы

Информация об исключительных ситуациях передается с помощью стандартного типа данных

Vector.h

#pragma once

#include <iostream>

#include <exception>

using namespace std;

class Vector

{

private:

int\* a; //массив

int size; //размер (сколько эл-тов добавлено)

int max\_size; //максим размер (сколько можно добавить элементов)

const int MAX\_SIZE = 30;

public:

Vector(int max\_size);

Vector(const Vector& v);

~Vector();

int get\_size() { return size; }

Vector& operator=(const Vector& v);

int& operator[](int index);

int operator()();

Vector operator\*(int k);

Vector& operator+(int k);

Vector& operator--();

class iterator

{

int\* current;

public:

iterator(int\* p) : current(p){}

int& operator\*() { return \*current; }

int& operator\*() const { return \*current; }

bool operator==(const iterator& it) const { return current == it.current; }

bool operator!=(const iterator& it) const { return current != it.current; }

iterator& operator++() { ++current; return \*this; }

iterator operator++(int) { int\* tmp = current; ++current; return iterator(tmp); }

iterator& operator-(int n)

{

current = current - n;

return \*this;

}

friend class Vector;

};

iterator begin() const { return iterator(a); }

iterator end() const { return iterator(a+size); }

};

Vector.cpp

#include "Vector.h"

Vector::Vector(int max\_size)

{

if (max\_size > MAX\_SIZE)

throw exception("размер больше максимального");

size = 0;

this->max\_size = max\_size;

a = new int[max\_size];

}

Vector::Vector(const Vector& v)

{

size = v.size;

max\_size = v.max\_size;

a = new int[max\_size];

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

{

a[i] = v.a[i];

}

}

Vector::~Vector()

{

delete[] a;

}

Vector& Vector::operator=(const Vector& v)

{

if(this != &v)

{

delete[] a;

a = new int[v.max\_size];

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

a[i] = v.a[i];

size = v.size;

max\_size = v.max\_size;

}

return \*this;

}

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

{

if (index < 0 || index >= size)

throw exception("выход за границы вектора");

return a[index];

}

int Vector::operator()()

{

return size;

}

Vector Vector::operator\*(int k)

{

Vector res(\*this);

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

{

res.a[i] = res.a[i] \* k;

}

return res;

}

Vector& Vector::operator+(int k)

{

if (size == max\_size)

throw exception("вектор полный");

a[size] = k;

size++;

return \*this;

}

Vector& Vector::operator--()

{

if (size == 0)

throw exception("вектор пустой");

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

{

a[i] = a[i + 1];

}

size--;

return \*this;

}

Main.cpp

#include <iostream>

#include "Vector.h"

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Размер вектора = ";

int size;

cin >> size;

try

{

Vector v(size);

do

{

cout << "1.Добавить элемент" << endl;

cout << "2.Удалить элемент из начала вектора" << endl;

cout << "3.Получить элемент по индексу" << endl;

cout << "4.Вывести элементы" << endl;

cout << "5.Умножение вектора на число" << endl;

cout << "6.Получение элемента с конца вектора" << endl;

cout << "0.Выход" << endl;

int oper;

cin >> oper;

if (oper == 0)

break;

try

{

switch (oper)

{

case 1:

{

cout << "Введите значение элемента: ";

int element;

cin >> element;

v = v + element;

break;

}

case 2:

{

--v;

break;

}

case 3:

{

cout << "Введите индекс элемента: ";

int index;

cin >> index;

cout << v[index] << endl;

break;

}

case 4:

{

Vector::iterator start = v.begin();

Vector::iterator end = v.end();

while (start != end)

{

cout << \*start << " ";

start++;

}

cout << endl;

break;

}

case 5:

{

cout << "Введите число: ";

int k;

cin >> k;

v = v \* k;

break;

}

case 6:

{

cout << "Введите номер: ";

int n;

cin >> n;

Vector::iterator end = v.end();

end = end - n;

cout << \*end << endl;

break;

}

default:

break;

}

}

catch (const exception & exc)

{

cout << exc.what() << endl;

}

}

while (true);

}

catch (const exception& exc)

{

cout << exc.what() << endl;

}

}

Информация об исключительных ситуациях передается с помощью пользовательского класса

Error.h

#pragma once

#include <iostream>

using namespace std;

class error

{

private:

string message;

public:

error(string message) : message(message){}

string get\_message() const { return message; }

};

Vector.h

#pragma once

#include <iostream>

#include <exception>

#include "error.h"

using namespace std;

class Vector

{

private:

int\* a; //массив

int size; //размер (сколько эл-тов добавлено)

int max\_size; //максим размер (сколько можно добавить элементов)

const int MAX\_SIZE = 30;

public:

Vector(int max\_size);

Vector(const Vector& v);

~Vector();

int get\_size() { return size; }

Vector& operator=(const Vector& v);

int& operator[](int index);

int operator()();

Vector operator\*(int k);

Vector& operator+(int k);

Vector& operator--();

class iterator

{

int\* current;

public:

iterator(int\* p) : current(p) {}

int& operator\*() { return \*current; }

int& operator\*() const { return \*current; }

bool operator==(const iterator& it) const { return current == it.current; }

bool operator!=(const iterator& it) const { return current != it.current; }

iterator& operator++() { ++current; return \*this; }

iterator operator++(int) { int\* tmp = current; ++current; return iterator(tmp); }

iterator& operator-(int n)

{

current = current - n;

return \*this;

}

friend class Vector;

};

iterator begin() const { return iterator(a); }

iterator end() const { return iterator(a + size); }

};

Vector.cpp

#include "Vector.h"

Vector::Vector(int max\_size)

{

if (max\_size > MAX\_SIZE)

throw error("размер больше максимального");

size = 0;

this->max\_size = max\_size;

a = new int[max\_size];

}

Vector::Vector(const Vector& v)

{

size = v.size;

max\_size = v.max\_size;

a = new int[max\_size];

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

{

a[i] = v.a[i];

}

}

Vector::~Vector()

{

delete[] a;

}

Vector& Vector::operator=(const Vector& v)

{

if (this != &v)

{

delete[] a;

a = new int[v.max\_size];

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

a[i] = v.a[i];

size = v.size;

max\_size = v.max\_size;

}

return \*this;

}

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

{

if (index < 0 || index >= size)

throw error("выход за границы вектора");

return a[index];

}

int Vector::operator()()

{

return size;

}

Vector Vector::operator\*(int k)

{

Vector res(\*this);

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

{

res.a[i] = res.a[i] \* k;

}

return res;

}

Vector& Vector::operator+(int k)

{

if (size == max\_size)

throw error("вектор полный");

a[size] = k;

size++;

return \*this;

}

Vector& Vector::operator--()

{

if (size == 0)

throw error("вектор пустой");

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

{

a[i] = a[i + 1];

}

size--;

return \*this;

}

Main.cpp

#include <iostream>

#include "Vector.h"

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Размер вектора = ";

int size;

cin >> size;

try

{

Vector v(size);

do

{

cout << "1.Добавить элемент" << endl;

cout << "2.Удалить элемент из начала вектора" << endl;

cout << "3.Получить элемент по индексу" << endl;

cout << "4.Вывести элементы" << endl;

cout << "5.Умножение вектора на число" << endl;

cout << "6.Получение элемента с конца вектора" << endl;

cout << "0.Выход" << endl;

int oper;

cin >> oper;

if (oper == 0)

break;

try

{

switch (oper)

{

case 1:

{

cout << "Введите значение элемента: ";

int element;

cin >> element;

v = v + element;

break;

}

case 2:

{

--v;

break;

}

case 3:

{

cout << "Введите индекс элемента: ";

int index;

cin >> index;

cout << v[index] << endl;

break;

}

case 4:

{

Vector::iterator start = v.begin();

Vector::iterator end = v.end();

while (start != end)

{

cout << \*start << " ";

start++;

}

cout << endl;

break;

}

case 5:

{

cout << "Введите число: ";

int k;

cin >> k;

v = v \* k;

break;

}

case 6:

{

cout << "Введите номер: ";

int n;

cin >> n;

Vector::iterator end = v.end();

end = end - n;

cout << \*end << endl;

break;

}

default:

break;

}

}

catch (const error & exc)

{

cout << exc.get\_message() << endl;

}

} while (true);

}

catch (const error & exc)

{

cout << exc.get\_message() << endl;

}

}

1. Результат выполнения

