Лабораторная работа №9

Обработка исключительных ситуаций

1. Цель задания:

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2) Разработка программы, обрабатывающей исключительные ситуации.

2. Теоретические сведения

2.1. Механизм обработки исключений.

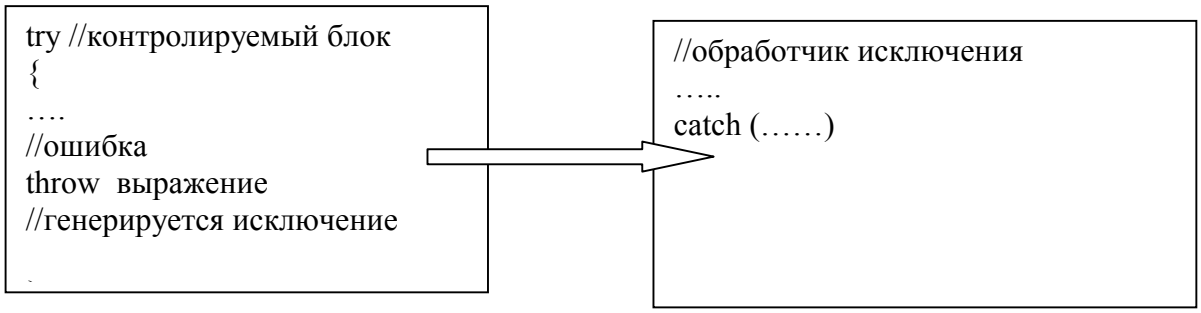
Исключение – это непредвиденное или аварийное событие.
В C++ исключение – это объект, который система должна генерировать при возникновении исключительной ситуации. Генерация такого объекта и создает исключительную ситуацию.
Исключения позволяют разделить вычислительный процесс на 2 части:

1) обнаружение аварийной ситуации (неизвестно как обрабатывать);
2) обработка аварийной ситуации (неизвестно, где она возникла).

Достоинства такого подхода:

1) удобно использовать в программе, которая состоит из нескольких модулей;
2) не требуется возвращать значение в вызывающую функцию

Общая схема:



Исключение генерируется оператором
`throw <выражение>,` где <выражение> -

- либо константа,
- либо переменная некоторого типа,
- либо выражение некоторого типа.

Тип объекта-исключения может быть как встроенным, так и определяемым пользователем. Для представления исключений часто используют пустой класс:

```
class ZeroDevide{};
class NegativeArg{};
```

Генерация исключения будет выглядеть:

```
throw ZeroDevide();//вызывается конструктор без параметров
```

или

```
throw new ZeroDevide();
```

Исключение надо перехватить и обработать. Для проверки возникновения исключения используется контролируемый блок `try{},` с которым связана одна или несколько секций-ловушек `catch`. Все переменные, объявленные в этом блоке, являются локальными для этого блока.

Форма записи секции-ловушки следующая:
`catch( спецификация исключения )`, где спецификация исключения может иметь три формы:

1) (тип имя)
2) (тип)
3) (…)

Тип – это встроенный тип или тип, определенный программистом.
Формы 1 и 2 обрабатывают конкретные исключения, а форма 3 перехватывает все исключения, такую ловушку надо помещать последней, тогда она будет обрабатывать все исключения, которые еще не были обработаны.
Форма 1 означает, что объект передается в блок обработки, чтобы его каким-то образом там использовать, например, для вывода информации в сообщении об ошибке.
Примеры:

```
catch( exception e) // по значению
catch( exception &e) // по ссылке
catch( const exception &e) // по константной ссылке
catch( exception *e) //по указателю
```

Лучше всего передавать объект по ссылке, т. к. при этом не создается временный объект-исключение.
Для каждой функции, метода, конструктора или деструктора можно в заголовке указать спецификацию исключений. Если в заголовке спецификация исключений не указана, считается, что функция может порождать любое исключение, если указана, то считается, что функция генерирует те исключения, которые явно указаны в этом списке.
Примеры:

```
void f1()throw(int,double);
void f2()throw(ZeroDivide);
```

Если спецификация имеет вид:

```
void f()throw();
```

то считается, что функция исключений не генерирует.
Наличие спецификаций исключения не является ограничением при реальной генерации исключений. Но если функция генерирует неспецифицированное исключение, то запускается стандартная функция `unexpected()`, которая вызывает функцию `terminate()`, что приводит к аварийному завершению программы.
При отсутствии подходящей секции-ловушки осуществляется вызов стандартной функции завершения `terminate()`. Эта функция вызывается из функции `unexpected()` при нарушении спецификации завершения.
Обе функции можно подменить собственными реализациями. Для этого необходимо
1. Подключить заголовок `#include <expection>`
2. Определить собственную функцию с прототипом `void F()` для подмены стандартной функции `terminate()`.
3. Указать имя этой функции в вызове функции `set_terminate(F);`

После этого вместо `terminate()` будет вызываться наша функция `F()`. Такая функция не должна возвращать управление оператором return или генерировать исключение `throw()`, она может только завершить программу функцией `exit()` или `abort()`.
Аналогично реализуется подмена стандартной функции unexpected():

```
set_ unexpected(F);
```

Функция может сгенерировать неспецифицированное исключение, в этом случае, если в спецификации исключений не указано исключение `bad_exception`, вызывается функция `terminate()`, в противном случае сгенерированное исключение подменяется на `bad_exception` и начинается поиск его обработчика.

2.2. Стандартные исключения.

В составе стандартной библиотеки C++ реализован ряд стандартных исключений, которые организованы в иерархию классов.
Эта иерархия может служить основой для создания собственных классов исключений и иерархии исключений. Можно определять собственные исключения, унаследовав их от класса `exception`.
Класс `exception` определен в стандартной библиотеке следующим образом:

```
class exception
{
public:
exception () throw();//конструктор без параметров
exception (const exception&) throw();//конструктор копирования
exception& operator= (const exception&) throw();//оператор =
virtual ~exception() throw();//деструктор
virtual const char*what() const throw();//генерирует сообщение
об ошибке
};
```

Все конструкторы и методы имеют спецификацию, запрещающую генерацию исключений.
Предполагается, что исключения типа

- `logic_error` - ошибки в логике программы, например, невыполнение какого-либо условия;
- `runtime_error` – ошибки возникают в результате непредвиденных обстоятельств при выполнении программы, например, переполнение при операциях с дробными числами;

Эти исключения программа должна генерировать самостоятельно оператором throw.
Пять стандартных исключений порождают различные механизмы C++.

- `bad_alloc` генерирует операция new, если не может быть выделена память
- `bad_cast и bad_typied` генерируются при динамической идентификации типов (RTTI)
- `ios_base::failure` генерируется системой ввода/вывода.
- `bad_exception` генерируется, если спецификация исключений содержит `bad_exception`.

3. Создание собственной иерархии исключений

Для создания собственной иерархии исключений надо объявить свой базовый класс-исключение, например:

```
class BaseException{};
```

Остальные классы будут наследниками этого класса, аналогично тому, как это сделано в иерархии стандартных исключений:

```
class Child_Exception1:public BaseException{};
class Child_Exception2:public BaseException{};
```

Класс `BaseException` можно унаследовать от стандартного класса `exception`

```
class BaseException: public exception{};
```

Наследование от стандартных классов позволит использовать метод `what` для вывода сообщений об ошибках.
Иерархия классов-исключений позволяет вместо нескольких разных блоков-ловушек написать единственный блок с типом аргумента базового класса.

3. Постановка задачи

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

4. Ход работы

Задача

Реализовать класс Вектор. Размер вектора ограничен значением MAX_SIZE=30.
Перегрузить для него операции

- доступ по индексу([int i]),
- добавление элемента (+ int),
- удаление элемента из начала вектора (--).

Предусмотреть генерацию исключительных ситуаций.

Исключительные ситуации генерируются:
1 – в конструкторе с параметром при попытке создать вектор больше максимального размера;
2, 3 – в операции [] – при попытке обратиться к элементу с номером меньше 0 или больше текущего размера вектора;
4 – в операции + – при попытке добавить элемент с номером больше максимального размера;
5 – в операции – при попытке удалить элемент из пустого вектора.
ВАРИАНТ РЕАЛИЗАЦИИ 1.
Информация об исключительных ситуациях передается с помощью стандартного типа данных.

1. Создать пустой проект.
2. Добавить в него класс Vector.
3. В файл Vector.h добавить описание класса Vector:

```
#pragma once
#include <iostream>
using namespace std;
const int MAX_SIZE=30;//максимальный размер вектора
class Vector
{
	int size;//текущий размер
	int *beg;//указатель на начало динамического массива
public:
	Vector(){size=0;beg=0;}//конструктор без параметров
	Vector(int s);//конструктор с параметром
	Vector(int s,int* mas);//конструктор с параметром
	Vector(const Vector&v);//конструктор копирования
	~Vector();//деструктор
	const Vector& operator=(const Vector&v);//операция присваивания
	int operator[](int i);//доступ по индексу
	Vector operator+(int a);//добавление элемента
	Vector operator--();//удаление элемента
//дружественные функции ввода-вывода
	friend ostream& operator<<(ostream&out,const Vector&v);
	friend istream& operator>>(istream& in, Vector&v);
};
```

4. В файл Vector.cpp добавить определение методов класса Vector:

```
Vector::Vector(int s)
{
	//если текущий размер больше максимального, то генерируется исключение
	if(s>MAX_SIZE) throw 1;
	size=s;
	beg=new int [s];
	for(int i=0;i<size;i++)
		beg[i]=0;
}
Vector::Vector(const Vector &v)
{
	size=v.size;
	beg=new int [size];
	for(int i=0;i<size;i++)
		beg[i]=v.beg[i];
}
Vector::~Vector()
{
	if (beg!=0) delete[]beg;
}
Vector::Vector(int s,int *mas)
{
//если текущий размер больше максимального, то генерируется исключение
	if(s>MAX_SIZE) throw 1;
	size=s;
	beg=new int[size];
	for(int i=0;i<size;i++)
		beg[i]=mas[i];
}
const Vector& Vector::operator =(const Vector &v)
{
	if(this==&v)return *this;
	if(beg!=0) delete []beg;
	size=v.size;
	beg=new int [size];
	for(int i=0;i<size;i++)
		beg[i]=v.beg[i];
	return*this;
}

ostream& operator<<(ostream&out, const Vector&v)
{
	if(v.size==0) out<<"Empty\n";
	else
	{
	for (int i=0;i<v.size;i++)
		out<<v.beg[i]<<" ";
	out<<endl;
	}
	return out;
}
istream& operator >>(istream&in, Vector&v)
{
	for(int i=0;i<v.size;i++)
	{
		cout<<">";
		in>>v.beg[i];
	}
	return in;
}
int Vector::operator [](int i)
{
	if(i<0)throw 2;//если индекс отрицательный, то генерируется исключение
	//если индекс больше размер вектора, то генерируется исключение
```

```
if(i>=size) throw error("Vector length more than MAXSIZE\n");
            return beg[i];
}
Vector Vector::operator +(int a)
{
//если при добавлении элемента размер вектора станет больше максимального,
//то генерируется исключение
      if(size+1==MAX_SIZE) throw 4;
      Vector temp(size+1,beg);
      temp.beg[size]=a;
      return temp;
}

Vector Vector::operator --()
{
//если вектор пустой, то удалить элемент нельзи и генерируется исключение
      if(size==0) throw 5;
      if (size==1)//если в вектор один элемент
      {
            size=0;
            delete[]beg;
            beg=0;
            return *this;
      };
      Vector temp(size,beg);
      delete[]beg;
      size--;
      beg=new int[size];
      for(int i=0;i<size;i++)
            beg[i]=temp.beg[i];
      return*this;

}
```

5. Добавить в проект файл lab9_main.cpp. В файл записать функцию main(), создающую объекты класса Vector и позволяющую генерировать исключительные ситуации.

```
#include "Vector.h"
#include <iostream>
using namespace std;
int main()
{
      //контролируемый блок
      try
      {
      Vector x(2);//вектор из двух элементов
      Vector y;//пустой вектор
      cout<<x;//печать вектора x
      cout<<"Nomer?";
      int i;
      cin>>i;
//вывод элемента с номером i, если номер больше 2 или меньше 0, то
//генерируется исключительная ситуация
      cout<<x[i]<<endl;
//добавление элемента в вектор, если MAX_SIZE=2, то генерируется
//исключительная ситуация
      y=x+3;
      cout<<y;
//удалить один элемент из вектора
      --x;
      cout<<x;
//удалить один элемент из вектора
      --x;
      cout<<x;//вектор пустой
//удалить один элемент из вектора
```

```
//генерируется исключительная ситуация

	--x;
	}
	//обработчик исключения
	catch(int)
	{cout<<"ERROR!!!"<<endl;}//сообщение об ошибке
	return 0;
}
```

6. Выполнить тестирование программы с генерацией различных исключительных ситуаций.

ВАРИАНТ РЕАЛИЗАЦИИ 2.

Информация об исключительных ситуациях передается с помощью пользовательского класса.

1. Создать пустой проект.
2. Добавить в него файл error.h.
3. В файл error.h добавить описание класса error:

```
#pragma once
#include <string>
#include <iostream>
using namespace std;
class error //класс ошибка
{
	string str;
public:
//конструктор, инициирует атрибут str сообщением об ошибке
	error(string s){str=s;}
	void what(){cout<<str<<endl;} //выводит значение атрибута str
};
```

4. Добавить класс Vector. В файл Vector.h добавить описание класса Vector:

```
#pragma once
#include <iostream>
using namespace std;
const int MAX_SIZE=20;
class Vector
{
	int size;
	int *beg;
public:
	Vector(){size=0;beg=0;}
	Vector(int s);
	Vector(int s,int* mas);
	Vector(const Vector&v);
	~Vector();
	const Vector& operator=(const Vector&v);
	int operator[](int i);
	Vector operator+(int a);
	Vector operator--();
	friend ostream& operator<<(ostream&out,const Vector&v);
	friend istream& operator>>(istream& in, Vector&v);
};
```

5. В файл Vector.cpp добавить определение методов класса Vector:

```
include "Vector.h"
#include "Error.h"
#include <iostream>
using namespace std;

Vector::Vector(int s)
```

```
{
	if(s>MAX_SIZE) throw error("Vector length more than MAXSIZE\n");
	size=s;
	beg=new int [s];
	for(int i=0;i<size;i++)
		beg[i]=0;
}
Vector::Vector(const Vector &v)
{
	size=v.size;
	beg=new int [size];
	for(int i=0;i<size;i++)
		beg[i]=v.beg[i];
}
Vector::~Vector()
{
	if (beg!=0) delete[]beg;
}
Vector::Vector(int s,int *mas)
{
	if(s>MAX_SIZE) throw error("Vector length more than MAXSIZE\n");
	size=s;
	beg=new int[size];
	for(int i=0;i<size;i++)
		beg[i]=mas[i];
}
const Vector& Vector::operator =(const Vector &v)
{
	if(this==&v)return *this;
	if(beg!=0) delete []beg;
	size=v.size;
	beg=new int [size];
	for(int i=0;i<size;i++)
		beg[i]=v.beg[i];
	return*this;
}

ostream& operator<<(ostream&out, const Vector&v)
{
	if(v.size==0) out<<"Empty\n";
	else
	{
	for (int i=0;i<v.size;i++)
		out<<v.beg[i]<<" ";
	out<<endl;
	}
	return out;
}
istream& operator >>(istream&in, Vector&v)
{
	for(int i=0;i<v.size;i++)
	{
		cout<<">";
		in>>v.beg[i];
	}
	return in;
}
int Vector::operator [](int i)
{
	if(i<0) throw error("index <0");
	if(i>=size) throw error("index>size");
	return beg[i];
}
Vector Vector::operator +(int a)
```

```
{
	if(size+1==MAX_SIZE) throw 4;
	Vector temp(size+1,beg);
	temp.beg[size]=a;
	return temp;
}

Vector Vector::operator --()
{
	if(size==0) throw error("Vector is empty");
	if (size==1)
	{
		size=0;
		delete[]beg;
		beg=0;
		return *this;
	};
	Vector temp(size,beg);
	delete[]beg;
	size--;
	beg=new int[size];
	for(int i=0;i<size;i++)
		beg[i]=temp.beg[i];
	return*this;

}
```

6. Добавить в проект файл lab9_main.cpp. В файл записать функцию main(), создающую объекты класса Vector и позволяющую генерировать исключительные ситуации.

```
#include "Vector.h"
#include "Error.h"
#include <iostream>
using namespace std;
int main()
{
	try
	{
	Vector x(2);
	Vector y;
	cout<<x;
	cout<<"Nomer?";
	int i;
	cin>>i;
	cout<<x[i]<<endl;
	y=x+3;
	cout<<y;
	--x;
	cout<<x;
	--x;
cout<<x;
	--x;
	}

	catch(error e)
	{e.what();}

	return 0;
}
```

7. Выполнить тестирование программы с генерацией различных исключительных ситуаций.

ВАРИАНТ РЕАЛИЗАЦИИ 3

Информация об исключительных ситуациях передается с помощью иерархии пользовательских классов.

1. Создать пустой проект.
2. Добавить в него файл error.h.
3. В файл error.h добавить описание иерархии пользовательских классов для определения исключительных ситуаций.

```
#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";}
	virtual void what(){cout<<msg<<msg_;}
};
class EmptySizeError:public SizeError //удаление из пустого вектора
{
protected:
	string msg_;
public:
	EmptySizeError(){SizeError();msg_="Vector is empty\n";}
	virtual void what(){cout<<msg<<msg_;}
};
class IndexError1:public IndexError //индекс меньше нуля
{
protected:
	string msg_;
public:
	IndexError1(){IndexError();msg_="index <0\n";}
	virtual void what(){cout<<msg<<msg_;}
};
class IndexError2:public IndexError //индекс больше текущего размера вектора
{
protected:
		string msg_;
public:
	IndexError2(){IndexError();msg_="index>size\n";}
	virtual void what(){cout<<msg<<msg_;}
};
```

4. Добавить класс Vector. В файл Vector.h добавить описание класса Vector:

```
const int MAX_SIZE=20;
class Vector
{
	int size;
	int *beg;
public:
	Vector(){size=0;beg=0;}
	Vector(int s);
	Vector(int s,int* mas);
	Vector(const Vector&v);
	~Vector();
	const Vector& operator=(const Vector&v);
	int operator[](int i);
	Vector operator+(int a);
	Vector operator--();
	friend ostream& operator<<(ostream&out,const Vector&v);
	friend istream& operator>>(istream& in, Vector&v);
};
```

5. В файл Vector.cpp добавить определение методов класса Vector:

```
#include "Vector.h"
#include "Error.h"
#include <iostream>
using namespace std;

Vector::Vector(int s)
{

	if(s>MAX_SIZE) throw MaxSizeError();
	size=s;
	beg=new int [s];
	for(int i=0;i<size;i++)
		beg[i]=0;
}
Vector::Vector(const Vector &v)
{
	size=v.size;
	beg=new int [size];
	for(int i=0;i<size;i++)
		beg[i]=v.beg[i];
}
Vector::~Vector()
{
	if (beg!=0) delete[]beg;
}
Vector::Vector(int s,int *mas)
{
	size=s;
	beg=new int[size];
	for(int i=0;i<size;i++)
		beg[i]=mas[i];
}
const Vector& Vector::operator =(const Vector &v)
{
	if(this==&v)return *this;
	if(beg!=0) delete []beg;
	size=v.size;
	beg=new int [size];
	for(int i=0;i<size;i++)
		beg[i]=v.beg[i];
	return*this;
```

```
}

ostream& operator<<(ostream&out, const Vector&v)
{
	if(v.size==0) out<<"Empty\n";
	else
	{
	for (int i=0;i<v.size;i++)
		out<<v.beg[i]<<" ";
	out<<endl;
	}
	return out;
}
istream& operator >>(istream&in, Vector&v)
{
	for(int i=0;i<v.size;i++)
	{
		cout<<">";
		in>>v.beg[i];
	}
	return in;
}
int Vector::operator [](int i)
{

	if(i<0)throw IndexError1();
	if(i>=size) throw IndexError2();
	return beg[i];
}

Vector Vector::operator +(int a)
{
	if(size+1==MAX_SIZE) throw MaxSizeError();
	Vector temp(size+1,beg);
	temp.beg[size]=a;
	return temp;
}
Vector Vector::operator --()
{
	if(size==0) throw EmptySizeError();
	if (size==1)
	{
		size=0;
		delete[]beg;
		beg=0;
		return *this;
	};
	Vector temp(size,beg);
	delete[]beg;
	size--;
	beg=new int[size];
	for(int i=0;i<size;i++)
		beg[i]=temp.beg[i];
	return*this;
}
```

6. Добавить в проект файл lab9_main.cpp. В файл записать функцию main(), создающую объекты класса Vector и позволяющую генерировать исключительные ситуации.

```
#include "Vector.h"
#include "Error.h"
#include <iostream>
using namespace std;
int main()
{
```

```
        try
        {
        Vector x(2);
        Vector y;
        cout<<x;
        cout<<"Nomer?";
        int i;
        cin>>i;
        cout<<x[i]<<endl;
        y=x+3;
        cout<<y;
        --x;
        cout<<x;
        --x;
        cout<<x;
        --x;
        }

              catch(Error &e)
        {
              e.what();
        }

        return 0;
}
```

7. Выполнить тестирование программы с генерацией различных исключительных ситуаций.

5. Варианты

№	Задание	Вариант реализации
1	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; + число – добавляет константу ко всем элементам вектора; - n- удаляет n элементов из конца вектора.	1 , 2
2	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: []– доступа по индексу; int() – определение размера вектора; - n – удаляет n элементов из конца вектора; + n - добавляет n элементов в конец вектора.	2 , 3
3	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; + +– добавляет элемент в вектор (постфиксная операция добавляет элемент в конец, префиксная в начало)	3 , 1
4	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу;	1 , 2

	() – определение размера вектора; - - – удаляет элемент из вектора (постфиксная операция удаляет элемент в конец вектора, префиксная – в начало)	
5	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера вектора; * вектор – умножение элементов векторов a[i]*b[i]; + n – переход вправо к элементу с номером n .	2 , 3
6	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера множества; + – объединение множеств; ++ - добавление элемента в множество .	3 , 1
7	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера вектора; * – пересечение множеств; -- - удаление элемента из множества.	1 , 2
8	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; == - проверка на равенство; > число – принадлежность числа множеству; - n - переход влево к элементу с номером n.	2 , 3
9	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; != - проверка на неравенство; < число – принадлежность числа множеству; + n – переход вправо к элементу с номером n .	3 , 1
10	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; - – разность множеств; -- – удаление элемента из множества.	1 , 2

11	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: []– доступа по индексу; int() – определение размера списка; + вектор – сложение элементов списков a[i]+b[i]; - n - переход влево к элементу с номером n.	2 , 3
12	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; + число – добавляет константу ко всем элементам вектора; ++ - добавление элемента в конец списка.	1 , 3
13	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; + вектор – добавление списка b к списку a (a+b) + число – добавляет элемент в начало списка;	1 , 2
14	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; () – определение размера списка; * число – умножает все элементы списка на число; - n – переход влево к элементу с номером n.	2 , 3
15	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера списка; * вектор – умножение элементов списков a[i]*b[i]; +n - переход вправо к элементу с номером n.	3 , 1

6. Контрольные вопросы

1. Что представляет собой исключение в C++?
2. На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?
3. Какой оператор используется для генерации исключительной ситуации?
4. Что представляет собой контролируемый блок? Для чего он нужен?
5. Что представляет собой секция-ловушка? Для чего она нужна?
6. Какие формы может иметь спецификация исключения в секции ловушке? В каких ситуациях используются эти формы?

7. Какой стандартный класс можно использовать для создания собственной иерархии исключений?
8. Каким образом можно создать собственную иерархию исключений?
9. Если спецификация исключений имеет вид: void f1()throw(int,double); то какие исключения может прождать функция f1()?
10. Если спецификация исключений имеет вид: void f1()throw(); то какие исключения может прождать функция f1()?
11. В какой части программы может генерироваться исключение?
12. Написать функцию, которая вычисляет площадь треугольника по трем сторонам (формула Герона).
    Функцию реализовать в 4 вариантах:
    - без спецификации исключений;
    - со спецификацией throw();
    - с конкретной спецификацией с подходящим стандартным исключением;
    - спецификация с собственным реализованным исключением.

7. Содержание отчета

1) Постановка задачи (общая и конкретного варианта).
2) Словесное описание исключительных ситуаций.
3) Определение класса error или иерархии пользовательских классов для определения исключительных ситуаций (если есть необходимость).
4) Описание класса-контейнера.
5) Определение компонентных функций для класса-конетейнера.
6) Функция main().
7) Объяснение результатов работы программы.
8) Ответы на контрольные вопросы.