# Лабораторная работа №6. Обработка ошибок, исключительные ситуации.

## Раскрутка стека

Из лекции Вы узнали, что при выбрасывании исключения последовательное выполнение программы прерывается и происходит процесс **раскрутки стека** (stack unwinding), при которой происходит уничтожение всех ранее созданных, но еще не разрушенных объектов в автоматической памяти и так до тех пор, пока не произойдет переход к соответствующему обработчику исключительной ситуации. В случае, если подходящего обработчика исключений найдено не будет, исключение будет перехвачено средой выполнения, что приведет к принудительному завершению программы.

Рассмотрим данный процесс более подробно. Стоит отметить, что раскрутка стека – процесс, не контролируемый программистом, однако мы можем понаблюдать за ним, воспользовавшись тем, что во время нее вызываются деструкторы созданных объектов. Для начала создадим специальный объект-наблюдатель, который будет сообщать нам о своем создании и разрушении:

```
class CObject
{
public:
   CObject(std::string const& objectName)
      :m_objectName(objectName)
   {
      std::cout << m_objectName << " is being created\n";
   }

   ~CObject()
   {
      std::cout << m_objectName << " is being destroyed\n";
   }
private:
```

```
   std::string const m_objectName;
};
```

Проверим наш объект на работоспособность:

```
int main(int argc, char * argv[])
{
   CObject object1("object1");
   {
      CObject object2("object2");
      CObject object3("object3");
   }
   CObject object4("object4");
   CObject *pObject5 = new CObject("pObject5");
   return 0;
}
```

Запустим программу и проанализируем выводимый ею результат

```
object1 is being created
object2 is being created
object3 is being created
object3 is being destroyed
object2 is being destroyed
object4 is being created
pObject5 is being created
object4 is being destroyed
object1 is being destroyed
```

Мы видим подтверждение того, что время жизни автоматической переменной прекращается, как только происходит выход из блока, в котором она объявлена. Кроме того, в C++ автоматические переменные уничтожаются в порядке, обратном их созданию.

Также видно, что объект **pObject5**, созданный в динамической памяти при помощи оператора **new**, не был уничтожен автоматически – для его уничтожения требуется оператор **delete**.

Далее, создадим еще один вспомогательный класс, который будет по такому же принципу сообщать нам о входе и выходе из функции:

```
class CFunctionCall
{
public:
   CFunctionCall(std::string const& functionName)
      :m_functionName(functionName)
   {
      std::cout << "Entering " << functionName << "\n";
   }
   ~CFunctionCall()
   {
      std::cout << "Exiting " << m_functionName << "\n";
   }
private:
   std::string const m_functionName;
};
```

Проверим класс в действии:

```
void SayHello()
{
   CFunctionCall sayHello("SayHello()");
   std::cout << "HELLO!\n";
}
```

```
int main(int argc, char * argv[])
{
   CFunctionCall main("main()");
   SayHello();
   return 0;
}
```

После запуска программы убеждаемся, что и класс CFunctionCall работает как положено:

```
Entering main()
Entering SayHello()
HELLO!
Exiting SayHello()
Exiting main()
```

Создадим собственный класс исключений, создание, копирование и разрушение которого также будем отслеживать в конструкторах и деструкторах:

```
class CMyException
{
public:
   CMyException()
      :m_copy(0)
   {
      std::cout << "CMyException is being created\n";
   }

   int GetCopyIndex()const
   {
      return m_copy;
   }

   // конструктор копирования будет создавать не совсем точную копию, т.к.
   // еще отслеживает и номер копии
   CMyException(CMyException const& e)
      :m_copy(e.m_copy + 1)
   {
      std::cout << "CMyException copy #" << m_copy << " is being created\n";
   }

   ~CMyException()
   {
      std::cout << "CMyException copy #" << m_copy << " is being destroyed\n";
   }
private:
   // номер копии
   int m_copy;
};
```

Проверим класс исключений в действии:

```
int main(int argc, char * argv[])
{
   try
   {
      throw CMyException();
   }
   catch (CMyException const& e)
   {
      std::cout << "Caught CMyException. Copy # is " << e.GetCopyIndex() << "\n";
   }
   return 0;
}
```

В данном случае копия объекта исключения создана не была:

```
CMyException is being created
Caught CMyException. Copy # is 0
CMyException copy #0 is being destroyed
```

Теперь проверим процесс раскрутки стека в действии:

```
void Function2()
{
   CFunctionCall fn("Function2()");

   CObject obj4("obj4");

   CObject * pobj5 = new CObject("pObj5");

   std::cout << "Throwing an exception\n";
   throw CMyException();
}

void Function1()
{
   CFunctionCall fn("Function1()");

   CObject obj2("obj2");
   {
      CObject obj3("obj3");
   }

   Function2();
}

int main(int argc, char * argv[])
{
   CFunctionCall fn("main()");

   try
   {
      CObject obj1("obj1");
      Function1();
   }
   catch (CMyException const& e)
   {
      std::cout << "CMyException is caught. Copy # " << e.GetCopyIndex() << "\n";
   }

   return 0;
}
```

Запустим программу:

```
Entering main()
obj1 is being created
Entering Function1()
obj2 is being created
obj3 is being created
obj3 is being destroyed
Entering Function2()
obj4 is being created
pObj5 is being created
Throwing an exception
CMyException is being created
obj4 is being destroyed
Exiting Function2()
obj2 is being destroyed
```

```
Exiting Function1()
obj1 is being destroyed
CMyException is caught. Copy index is 0
CMyException copy #0 is being destroyed
Exiting main()
```

Итак, что же произошло в процессе раскрутки стека сразу после выброса исключения?

1. Создался объект исключения CMyException
2. Произошло разрушение переменной obj4, находящейся в функции Function2
3. Произошел возврат из функции Function2 в вызвавшую ее функцию Function1
4. Произошло разрушение еще не разрушенной переменной obj2 в функции Function1. Отметим, что переменная obj3 разрушилась еще раньше – при выходе из блока, в котором была объявлена
5. Произошел возврат из функции Function1 обратно в функцию main()
6. Произошло разрушение объекта obj1, объявленного внутри блока try.
7. Произошел переход внутрь обработчика исключения CMyException
8. При выходе из обработчика произошло разрушение объекта исключения CMyException
9. По оператору return произошел выход из функции main().

## Исключения в списке инициализации конструктора

Иногда может потребоваться необходимость перехватить исключение, возникающее при инициализации полей класса. Разместить обработчик в теле конструктора не выйдет, т.к. он выполняется уже после того, как были проинициализированы все поля класса.

Специально для решения данной проблемы в Стандарт языка C++ был внесен специальный блок try-catch, являющийся телом функции, имеющий название **function-try-block**, что можно перевести как «контролируемый блок-функция».

Пример использования такого блока:

```
double MySqrt(double arg)
{
   if (arg < 0)
   {
      throw std::invalid_argument("Argument must not be negative");
   }
   return sqrt(arg);
}

class CSomeClass
{
public:
   CSomeClass(double arg)
   try
   :m_value(MySqrt(arg))
   {
      // тело конструктора
   }
   catch(...)
   {
      /*
      В тело конструктора мы уже не попадем,
      объект сконструировать у нас уже не получилось.
      Поэтому попробуем сделать хоть что-то
      */
      std::cout << "Unable to construct CSomeClass instance\n";

      // и перевыбрасываем пойманное исключение
      // (также можно выбросить другое, но главное - выбросить хоть что-то)
      throw;
```

```
    }

private:
   double m_value;
};

int main(int argc, char * argv[])
{
   try
   {
      CSomeClass someObject(-1);
   }
   catch (std::exception const& e)
   {
      std::cout << e.what() << "\n";
   }

   return 0;
}
```

Контролируемый блок-функцию можно объявить не только в конструкторе, но и в любом методе класса или функции. Например, для перехвата всех исключений нашего приложения можно записать следующий вариант функции main():

```
int main(int argc, char * argv[])
try
{
   // тело функции main
   // ...
   return 0;
}
catch (std::exception const& e)
{
   std::cout << "Error: " << e.what() << "\n";
   return 1;
}
catch (...)
{
   std::cout << "Unknown exception has been caught\n";
   return 2;
}
```

## Практические задания

На оценку «**удовлетворительно**» необходимо набрать **не менее 80 баллов**.

На оценку «**хорошо**» необходимо набрать **не менее 220 баллов**.

На оценку «**отлично**» необходимо набрать **не менее 350 баллов**.

### Обязательные задания

#### *Задание 1*

**Вариант 1 – Triangle – 30 баллов**

Разработайте класс CTriangle, представляющий треугольник, заданный длинами трех своих сторон и предоставляющий информацию о своих сторонах, площади и периметре.

Площадь треугольника, длины сторон которого известны, можно вычислить по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где a, b и c – длины сторон треугольника, а p – полупериметр:

$$p = \frac{(a + b + c)}{2}$$

Каркас класса представлен ниже:

```
class CTriangle
{
public:
	CTriangle(double side1, double side2, double side3);
	double GetSide1()const;
	double GetSide2()const;
	double GetSide3()const;
	double GetArea()const;
	double GetPerimeter()const;
};
```

При работе данного класса возможны следующие внештатные ситуации:

- Некорректное значение аргументов конструктора – длины сторон треугольника не могут быть отрицательными
- Не любые 3 отрезка могут сформировать треугольник.
  - В невырожденном треугольнике длина каждой стороны должна быть меньше суммы двух других сторон.
  - В случае вырожденного треугольника допускается равенство длины некоторой стороны треугольника сумме длин двух других сторон.
  - Если длина одного из отрезков превышает сумму двух других, то треугольник с их помощью создать не удастся.

В конструкторе класса CTriangle необходимо обрабатывать данные внештатные ситуации, выбрасывая соответствующие исключения. При некорректном значении аргументов необходимо выбрасывать исключение std::invalid_argument, а при недопустимой комбинации длин сторон – исключение std::domain_error. При создании исключения необходимо инициализировать его текстом сообщения об ошибке.

Разработайте на базе данного класса приложение, выполняющее считывание со стандартного потока ввода длины сторон трех отрезков, формирование из них треугольника и выводящее в стандартный поток вывода его площадь и периметр. Приложение должно перехватывать выбрасываемые исключения и выводить в поток вывода диагностическую информацию об исключительной ситуации (текст сообщения об ошибке из перехваченного сообщения).

Ввод данных прекращается, как только во входном потоке будет встречен символ конца файла.

**Для класса CTriangle должен быть разработан набор автоматических тестов, проверяющих работу его методов и конструктора как на корректных, так и некорректных наборах данных. Рекомендуется использование библиотек и фреймворков для написания тестов, например, boost::test, Google Test, CxxTest и других. Без тестов работа будет принята с коэффициентом не выше 0.3.**

### Вариант 2 – Решение квадратных уравнений – 30 баллов

Разработайте функцию **Solve**, выполняющую вычисление корней квадратного уравнения $ax^2 + bx + c = 0$ и возвращающую результат в виде структуры EquationRoots:

```
struct EquationRoots
{
	int numRoots;
```

```
	double roots[2];
};

// Вычисляем корни квадратного уравнения ax^2 + bx + c = 0
EquationRoots Solve(double a, double b, double c)
{
...
}
```

Функция Solve должна обрабатывать следующие внештатные ситуации:

- Не допускается нулевое значение коэффициента при $x^2$. В случае возникновения данной ситуации должно выбрасываться исключение **std::invalid_argument**.
- Квадратное уравнение может не иметь действительных корней. В случае возникновения данной ситуации должно выбрасываться исключение **std::domain_error**.

При выбрасывании исключения необходимо инициализировать его текстом с описанием проблемы.

Разработать на основе данной функции программу, запрашивающую со стандартного потока ввода коэффициенты квадратного уравнения и вычисление при помощи функции Solve корней квадратного уравнения с последующим выводом результата в стандартный поток вывода. Исключения, выбрасываемые функцией Solve, должны перехватываться программой, а текст диагностического сообщения с информацией об ошибке выводиться в стандартный поток вывода.

Ввод данных программой со стандартного ввода прекращается, как только будет встречен символ конца файла.

**Для функции Solve должен быть разработан набор автоматических тестов, ее работу как на корректных, так и некорректных наборах данных. Рекомендуется использование библиотек и фреймворков для написания тестов, например, boost::test, Google Test, CxxTest и других. Без тестов работа будет принята с коэффициентом не выше 0.3.**

### Вариант 3 – Решение кубических уравнений – 60 баллов

Разработайте функцию Solve3, выполняющую вычисление корней кубического уравнения $ax^3 + bx^2 + cx + d = 0$ и возвращающую результат в виде структуры EquationRoots3:

```
struct EquationRoots3
{
	int numRoots;
	double roots[3];
};

// Вычисляем корни кубического уравнения ax^3 + bx^2 + cx + d = 0
EquationRoots3 Solve3(double a, double b, double c, double d)
{
...
}
```

Функция Solve3 должна обрабатывать следующие внештатные ситуации:

- Не допускается нулевое значение коэффициента при $x^3$. В случае возникновения данной ситуации должно выбрасываться исключение std::invalid_argument.

При выбрасывании исключения необходимо инициализировать его текстом с описанием проблемы.

Разработать на основе данной функции программу, запрашивающую со стандартного потока ввода коэффициенты кубического уравнения и вычисление при помощи функции Solve3 корней кубического уравнения с последующим выводом результата в стандартный поток вывода. Исключения, выбрасываемые функцией Solve3, должны перехватываться программой, а текст диагностического сообщения с информацией об ошибке выводиться в стандартный поток вывода.

Ввод данных программой со стандартного ввода прекращается, как только будет встречен символ конца файла.

**Для функции Solve3 должен быть разработан набор автоматических тестов, ее работу как на корректных, так и некорректных наборах данных. Рекомендуется использование библиотек и фреймворков для написания тестов, например, boost::test, Google Test, CxxTest и других. Без тестов работа будет принята с коэффициентом не выше 0.3.**

### Вариант 4 – Решение уравнений 4 степени – 100 баллов

Разработайте функцию Solve4, выполняющую вычисление корней уравнения 4 степени $ax^4 + bx^3 + cx^2 + dx + e = 0$ и возвращающую результат в виде структуры EquationRoot4:

```
struct EquationRoot4
{
      Double numRoots
      double roots[4];
};

// Вычисляем корни кубического уравнения ax^4 + bx^3 + cx^2 + dx + e = 0
EquationRoot4 Solve4(double a, double b, double c, double d, double e)
{
...
}
```

Функция Solve4 должна обрабатывать следующие внештатные ситуации:

- Не допускается нулевое значение коэффициента при $x^4$. В случае возникновения данной ситуации должно выбрасываться исключение std::invalid_argument.
- Уравнение 4 степени может не иметь действительных корней. В случае возникновения данной ситуации должно выбрасываться исключение std::domain_error.

При выбрасывании исключения необходимо инициализировать его текстом с описанием проблемы.

Разработать на основе данной функции программу, запрашивающую со стандартного потока ввода коэффициенты уравнения 4 степени и вычисление при помощи функции Solve4 корней данного уравнения с последующим выводом результата в стандартный поток вывода. Исключения, выбрасываемые функцией Solve4, должны перехватываться программой, а текст диагностического сообщения с информацией об ошибке выводиться в стандартный поток вывода.

Ввод данных программой со стандартного ввода прекращается, как только будет встречен символ конца файла.

**Для функции Solve4 должен быть разработан набор автоматических тестов, ее работу как на корректных, так и некорректных наборах данных. Рекомендуется использование библиотек и фреймворков для написания тестов, например, boost::test, Google Test, CxxTest и других. Без тестов работа будет принята с коэффициентом не выше 0.3.**

### Вариант 5 – Сортировка строк - 20 баллов

Разработайте функцию **SortStrings3**, выполняющую упорядочивание трех строк в лексикографическом порядке и предоставляющую **гарантию отсутствия исключений**.

```
void SortStrings(std::string & s1, std::string & s2, std::string & s3);
```

На основе данной функции разработайте программу, считывающую строки со стандартного потока ввода и выводящую в стандартный поток вывода каждую тройку строк, отсортированных в лексикографическом порядке.

### Вариант 6 – Студент – 50 баллов

Реализуйте класс **CStudent**, описание методов которого представлено ниже. Все методы класса должны предоставлять строгую гарантию безопасности исключений, т.е. поддерживать семантику выполнения commit-or-rollback.

Студент характеризуется фамилией, именем, отчеством, а также возрастом. Имя и фамилия студента должны быть непустыми строками, содержащими символы, отличные от пробела. Отчество студента может быть пустой строкой (если отчество не пустое, то оно также должно содержать символы, отличные от пробела). Возраст студента может быть в диапазоне от 14 до 60 лет (включительно)

| Метод | Описание |
|---|---|
| Конструктор | Создает и инициализирует объект «Студент»,= с использованием указанного имени (name), фамилии (surname) и отчества (patronymic), а также возраста. В качестве отчества допускается пустая строка.<br>При недопустимом ФИО должно выбрасываться исключение std::invalid_argument. При недопустимом возрасте должно выбрасываться исключение std::out_of_range |
| GetName | Возвращает имя студента |
| GetSurname | Возвращает фамилию студента |
| GetPatronymic | Возвращает отчество студента |
| GetAge | Возвращает возраст студента |
| Rename(name, surname, patronymic) | Переименовывает студента. На ФИО налагаются ограничения, описанные выше. Не забудьте обеспечить семантику выполнения commit-or-rollback при выбрасывании исключения (в том числе и std::bad_alloc при нехватке памяти) |
| SetAge(age) | Изменяет возраст студента. Возраст не может быть изменен в сторону уменьшения (при нарушении бросать std::domain_error).<br>При выходе за пределы диапазона от 14 до 60, выбрасывать исключение std::out_of_range |

**Для класса CStudent должен быть разработан набор автоматических тестов, проверяющих работу его методов и конструктора как на корректных, так и некорректных наборах данных. Рекомендуется использование библиотек и фреймворков для написания тестов, например, boost::test, Google Test, CxxTest и других. Без тестов работа будет принята с коэффициентом не выше 0.3.**

### Вариант 7 – HTTP URL – 100 баллов

Разработайте класс CHttpUrl, выполняющий хранение http и https – URL-ов. В конструкторе класса должны осуществляться проверка валидности входных параметров.

Каркас класса:

```
enum Protocol
{
   HTTP,
   HTTPS
};
```

```
class CUrlParsingError : public std::invalid_argument
{
public:
  …
};

class CHttpUrl
{
public:
  // выполняет парсинг строкового представления URL-а, в случае ошибки парсинга
  // выбрасыват исключение CUrlParsingError, содержащее текстовое описание ошибки
  CHttpUrl(std::string const& url);

  /* инициализирует URL на основе переданных параметров.
    При недопустимости входных параметров выбрасывает исключение
    std::invalid_argument
    Если имя документа не начинается с символа /, то добавляет / к имени документа
  */
  CHttpUrl(
    std::string const& domain,
    std::string const& document,
    Protocol protocol = HTTP);

  /* инициализирует URL на основе переданных параметров.
    При недопустимости входных параметров выбрасывает исключение
    std::invalid_argument
    Если имя документа не начинается с символа /, то добавляет / к имени документа
  */
  CHttpUrl(
    std::string const& domain,
    std::string const& document,
    Protocol protocol,
    Unsigned short port);

  // возвращает строковое представление URL-а. Порт, являющийся стандартным для
  // выбранного протокола (80 для http и 443 для https) в URL не должен включаться
  std::string GetURL()const;

  // возвращает доменное имя
  std::string GetDomain()const;

  /*
    Возвращает имя документа. Примеры:
      /
      /index.html
      /images/photo.jpg
  */
  std::string GetDocument()const;

  // возвращает тип протокола
  Protocol GetProtocol()const;

  // возвращает номер порта
  unsigned short GetPort()const;
};
```

**Классы CUrlParsingError и CHttpUrl должны располагаться в разных парах .h/.cpp файлов**.

Конструктор класса должен анализировать входные параметры и выбрасывать нужный тип исключений в случае ошибок.

На основе данного класса напишите приложение, считывающее со стандартного потока ввода URL-ы и выводящее информацию об URL-е в стандартный поток вывода, пока не встретит символ конца файла.

Исключения, выбрасываемые классом должны перехватываться программой, и в стандартный поток вывода должна выводиться информация об ошибке обработки URL-а.

**Для класса CHttpUrl должен быть разработан набор автоматических тестов, проверяющих работу его методов и конструктора как на корректных, так и некорректных наборах данных. Рекомендуется использование библиотек и фреймворков для написания тестов, например, boost::test, Google Test, CxxTest и других. Без тестов работа будет принята с коэффициентом не выше 0.3.**

## Дополнительные задания

### *Задание 2*

#### Вариант 1 – StringStack - 150 баллов

Реализуйте класс **CStringStack**, реализующий стек строк. Для хранения данных стек должен использовать **динамический массив** (реализовать своими силами), либо односвязный список. Определите набор непредвиденных ситуаций и соответствующие им исключения, которые должны выбрасываться классом в случае ошибок. Методы класса должны предоставлять **уровень безопасности исключений не ниже строгого**, а деструктор – гарантировать отсутствие исключений. На защите лабораторной работы необходимо обосновать предоставление классом данных гарантий безопасности исключений.

**Для класса CStringStack должен быть разработан набор автоматических тестов, проверяющих работу его методов и конструктора как на корректных, так и некорректных наборах данных. Рекомендуется использование библиотек и фреймворков для написания тестов, например, boost::test, Google Test, CxxTest и других. Без тестов работа будет принята с коэффициентом не выше 0.3.**

#### Вариант 2 – StringList - 300 баллов

Реализуйте класс CStringList, реализующий двусвязный список строк. Список должен предоставлять следующие операции:

- Добавление строки в начало и в конец списка (за время O(1))
- Узнать количество элементов (за время O(1)), а также узнать пуст список или нет
- Удалить все элементы из списка (за время O(N) с использованием памяти O(1) в области стека)
- Вставка элемента в позицию, задаваемую итератором (за время O(1))
- Удаление элемента в позиции, задаваемой итератором (за время O(1))
- Получение итераторов, указывающих на начало списка и его конец (константные и некостантные), совместимых со алгоритмами STL и range-based for
- Реверсированные итераторы (константные и неконстантные)

Все операции над списком должны предоставлять уровень безопасности исключений не ниже строгого, а деструктор гарантировать отсутствие исключений.

Разрешается использовать умные указатели для хранения структуры данных списка. **Убедиться, что деструктор класса и метод Clear не используют неявным образом рекурсию** (в противном случае удаление списка с большим количеством элементов приведет к переполнению стека).

**Для класса CStringList должен быть разработан набор автоматических тестов, проверяющих работу его методов и конструктора как на корректных, так и некорректных наборах данных. Рекомендуется использование библиотек и фреймворков для написания тестов, например, boost::test, Google Test, CxxTest и других. Без тестов работа будет принята с коэффициентом не выше 0.3.**