# Информатика: семестр 3

Программирование на C++

Конспекты лекций

Лектор: Хирьянов Т.Ф.

28 ноября 2017 г.

# Оглавление

# Лекция 13

# Исключения в C++

## 13.1 Понятие исключения

Пусть некая функция **main()** вызывает функции **A,B,C,D** по схеме $A \to B \to C \to D$. Что произойдет, если одна из них не сможет выполнить свои обязательства (встретится какая-нибудь ошибка, ресурсы будут недоступны и т.п.)?

### Интерфейс функции

Прежде чем ответить на этот вопрос, разберемся с таким понятием, как *интерфейс функции*. Интерфейс функции – это некий абстрактный набор, содержащий характеризующую функцию информацию. Он включает *имя, смысл, типы параметров, их допустимые значения и возможные исключения*. Интерфейс функции должен быть продуман перед разработкой, на этапе осмысливания архитектуры целого приложения.

## 13.2 Исключения в C++

В C++, как и во многих языках программирования, наша ошибка выполнения будет обработана следующим образом: неисправная функция создаст **исключение** и вернет вызвавшей ее функции это исключение. Та, с свою очередь, должна будет его обработать дальше. Исключения, порождаемые функцией, являются самостоятельными типами. Они имеют свои поля и атрибуты.

## 13.3 Стек вызова

Когда одна функция вызывает другую, вторая отправляется в *стек вызовов*. Это такая очередь, содержащая функции в том порядке, в каком они вызывались. В стеке лежат функции и *адреса возврата* – адрес того места функции в оперативной памяти,

где была вызвана новая функция (чтобы впоследствии вернуться строго в то место, где выполнения программы прервались на вызов другой функции). Помимо адресов в стек вызовов уходят локальные переменные, место для аргументов функции, место для возвращаемого значения (или для адреса, если возвращаемое значение лежит в динамической памяти). При этом передаваемые аргументы будут передаваться по обычным правилам (с использованием конструктора копирования для объектов и т.п.).

## 13.4 Порядок вылета из программы

Что произойдет, если вылетит исключение и не будет обработано? Программа вылетит, причем весь код дальше выполнен не будет. Это может привести к утечке памяти (если мы таким способом пропустим **delete**, освобождающий память) и другим проблемам (о них ниже).

## 13.5 Реализация исключений в C++

Теперь посмотрим, какова реализация механизма исключений в C++. Работа с опасным кодом в этом языке осуществляется следующим образом:

Код 13.1: Синтаксис исключений

```
try {
    // Dangerous code
} catch(ExceptionType e) {
    // e -- exception variable, contains exception info
}
```

При этом **ExceptionType** – это тип исключения (см. 13.2). Если в опасный код выбросит исключение типа **ExceptionType**, то программа начнет выполнять код в блоке **catch**. Если это исключение другого типа, оно не будет обработано.

## 13.6 Проблемы исключений

В процессе 'вылета' из-за исключения остальные объекты нашей программы будут удалены. А для этого нужно вызвать их деструкторы. Что произойдет, если при этом деструктор сам выбросит исключение, не знает даже Т.Ф., поэтому деструкторы надо писать максимально простыми и не выбрасывающими исключения.

Еще одна проблема, которая может случиться – исключение во время выполнения конструктора. При таком исходе объект не будет до конца создан и для него не будет вызван деструктор. Это тоже может вести к утечкам памяти.

## 13.7 Иерархия исключений

Сами исключения, как объекты классов, имеют свою иерархию. Самое базовое исключение есть экземпляр класса **BaseException**. От него наследуются уже специфичные исключения, например, **ArithmeticException** – исключение в арифметике (деление на ноль и т.п.). Все эти типы можно передать в **catch**, тем самым указав типа ошибки, который будет обработан в конкретном блоке.

### Default exception

Исключение вида:

Код 13.2: Default exception

```
1 catch(...) {
2     // Your code
3 }
```

является исключением по умолчанию. Это, по-простому говоря, исключение на все случаи жизни. Т.Ф. не рекомендует им пользоваться, ведь этот тип практически ничего не будет знать о деталях ошибки.

## 13.8 Вторичные исключения

При обработке исключения в блоке **catch** может случиться такое, что с какими-то проблемами наш блок будет в состоянии разобраться, а с какими-то – нет. В последнем случае **catch** может сам выкинуть exception и передать его уже выше. Это делается командой **throw**.

Код 13.3: Пример вторичного исключения

```
1  try {
2      // dangerous code
3  } catch(SpecialEx e) {
4      if(canHandle()) {
5          // handle it
6          e.what()     // return string that tells about error
7      } else {
8          throw NewExceptionType;
9      }
10 }
```