

Лекция 10. Исключения.



НИЖЕГОРОДСКИЙ ИНСТИТУТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

НИИТ

Исключения (exception)

- Генерация сообщения об ошибке (throw)
- Перехват этих сообщений (catch)
- В программе может одновременно существовать только одно исключение.

throw и catch

```
class Stack {
public:
    Stack(int size);
    void push(char c);
    //...
};

class Overflow {};

class WrongSize {
public:
    int wsize;
    Wrong_size(int i):
        wsize(i) {}
};
```

```
void Stack::push(char c) {
    if (top<maxSize)
        storage[top++] = c;
    else
        throw Overflow();
}
```

```
void f() {
    try {
        Stack s(10);
        s.push('a');
    } catch (Overflow ex)
    {
        cerr << "Stack overflow";
    }
}
```

Выбор исключений

```
Stack::Stack(int size)
{
    if ( size > 10000) {
        throw WrongSize(size);
    } //...
}
```

```
char Stack::pop()
{
    if (top==0)
        throw Underflow();

    return storage[--top];
}
```

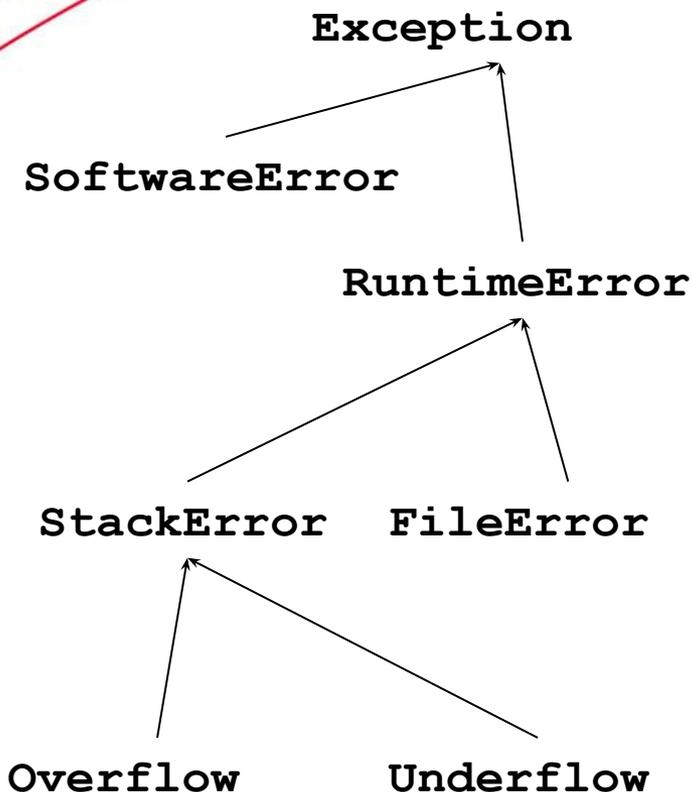
```
void f(unsigned int size)
{
    try {
        Stack s(size);
        s.push('a');
        char c = s.pop();
        char d = s.pop();
    }
    catch (WrongSize ws) {
        cerr << "Wrong size:" << ws.wsize;
    }
    catch (Overflow) { /*...*/
    }
    catch (Underflow) { /*...*/
    }
}
```

Группировка исключений

```
class Exception {};  
  
class StackError: public Exception {};  
  
class Overflow: public StackError {};  
class Underflow: public StackError {};  
class WrongSize: public StackError {};
```

```
try {  
    Stack s(size);  
    // ...  
}  
  
catch (WrongSize size_exc) {  
    // process wrongsize exception  
}  
  
catch (StackError) {  
    // general processing  
}
```

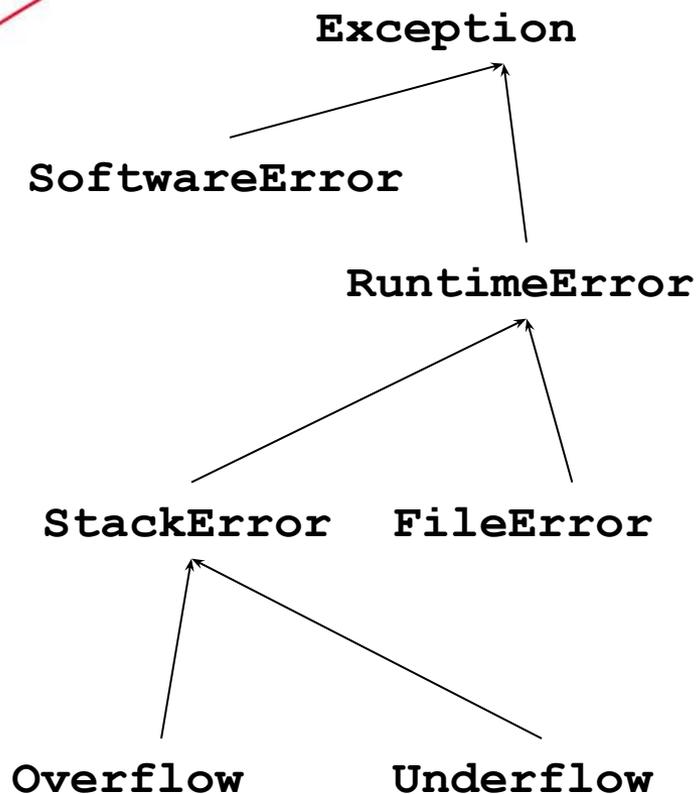
Перехват исключений



```

try {
    //...
}
catch (StackError& se) {
    // process Stack Error
}
catch (RuntimeError& ps) {
    // process Runtime Error
}
catch (Exception) {
    // process Any Internal Exception
}
catch (...) {
    // process any other exception
}
  
```

Перехват исключений 2



```

try {
    //...
}
catch (...) {
    // Все исключения перехватываются
    // здесь
}
catch (Exception* ex) {
    // process Any Internal Exception
}
catch (RuntimeError* re) {
    // process Runtime Error
}
catch (StackError* se) {
    // process Stack Error
}
  
```

Повторная генерация

```
void f()  
{  
    try {  
        // ...  
        throw Underflow();  
    }  
    catch (RuntimeError& re) {  
        if ( can_handle_it_completely(re) ) {  
            // process the exception here  
            return;  
        }  
        else {  
            do_what_you_can_do(re);  
            throw;  
        }  
    }  
}
```

Повторная генерация 2

```
void g()  
{  
    try {  
        f();  
    }  
    catch (StackError& re) {  
        // process stack error  
    }  
    catch (FileError& re) {  
        // process file error  
    }  
}
```

Исключения в конструкторах

Классические подходы:

- Возвратить объект в «неправильном» состоянии
- Присвоить значение глобальной переменной
- Использовать функцию инициализации
- Осуществлять инициализацию при первом вызове функции-члена

Исключения в конструкторах

```
Stack::Stack(int i)
{
    if ( (i < MIN_SIZE) ||
         (i > MAX_SIZE) )
    {
        throw WrongSize(i);
    }
    //...
}
```

```
Stack* get_stack(int size)
{
    try {
        Stack* s = new Stack(size);
        //...
        return s;
    }
    catch (WrongSize) {
        // handle bad stack size
    }
}
```

**Объект не создан, пока не завершится
выполнение его конструктора**
Nortel Networks Confidential

Исключения и инициализация членов

```
Schedule::Schedule(int i, Date d)
try
    : m_stack(i),
      m_date(d)
{
    // constructor
}
catch (Stack::Bad_Size) {
    // handle bad size of the stack member
}
catch (Date::Bad_Date) {
    // handle bad date of the date member
}
```

Исключения и копирование

- Копирующий *конструктор* подобен другим конструкторам:
 - может генерировать исключения
 - при этом объект не создается
- Копирующее *присваивание* перед генерацией исключения должно убедиться, что оба операнда находятся в корректном состоянии

Исключения в деструкторах

1. Нормальный вызов деструктора
2. Вызов в процессе обработки исключения
 - Во 2ом случае исключение не должно покинуть деструктор (`uncaught_exception()`)
 - Если все таки покинет вызывается `std::terminate()`
 - Т.е. если собираетесь бросить исключения в деструкторе, всегда пользуйтесь `uncaught_exception()` для проверки на уже существующее исключение.

```
typedef void (*terminate_handler) ();  
terminate_handler set_terminate(terminate_handler);
```

Конец