

Технология ООП

Санкт-Петербургский государственный политехнический университет

4 октября 2011

При обработке рассматриваются только ситуации внутреннего характера (нет памяти в области heap, не найден файл, переполнение и т.д.). Ситуация созданная нажатием Ctrl-C считается внешней.

Обработка исключительных ситуаций

Синтаксис исключений:

```
try {  
    ...  
}
```

Обозначает контролируемый блок кода.

Обработка исключительных ситуаций

Генерация исключений:

```
throw [ выражение ];
```

Обработка исключений:

```
catch(тип имя) { /*тело обработчика*/ }  
catch(тип) { /*тело обработчика*/ }  
catch(...) { /*тело обработчика*/ }
```

Обработка исключительных ситуаций

Обработка исключений:

```
catch(int i) { //int }  
catch(const char*) { //const char* }  
catch(Overflow) { //класс Overflow }  
catch(...) { /*обработка всех исключений*/ } 
```

Обработка исключительных ситуаций

После обработки исключения управление передается первому оператору находящемуся за блоком исключений. Туда же, минуя код всех обработчиков, передается управление, если исключение в try-блоке не было сгенерировано.

При генерации исключения в C++ происходят следующие действия:

- создается копия параметра `throw` в виде объекта, который существует до тех пор, пока исключение не будет обработано;
- вызываются деструкторы объектов выходящих из области действия;
- передается объект и управление обработчику имеющему совместимый тип с этим объектом.

При генерации исключений:

- создается копия параметра `throw` в виде объекта, который существует до тех пор, пока исключение не будет обработано;
- вызываются деструкторы объектов выходящих из области действия;
- передается объект и управление обработчику имеющему совместимый тип с этим объектом.

Обработчик считается найденным, если:

- тип объекта в обработке тот же, что и указан после `throw`, т.е. `T`, `const T`, `T&` или `const T&`, где `T` — тип исключения;
- является производным от указанного в параметре `catch`, если наследования производилось с ключем `public`;
- является указателем который может быть преобразован к нужному типу, например `void*` .

Если происходит вызов непредусмотренного исключения, то вызывается функция `unexpected()`, реализацию которой можно заменить при помощи `set_unexpected()`, если такой функции нету, то вызывается функция `terminate()`, реализацию которой можно заменить при помощи `set_terminate()`, если такой функции нету то происходит вызов функции `abort()`.

Обработка исключительных ситуаций

Типы исключений которые может генерировать функция, перечисляются после ключевого слова `throw` после прототипа функции.

```
void f1() throw (int , const char*) {/* может  
генерировать только типов int или char* */  
void f2() throw (Oops*) {/* исключения типа  
указатель на класс */}
```

Обработка исключительных ситуаций

```
void f1() throw ()  
{  
    //Тело функции, не порождающей исключений  
}
```

При переопределении виртуальной функции можно задавать список исключений такой же или более ограниченный чем в базовом классе.

Если функция вызывает не описанное исключение, вызывается функция `unexpected()`.

Исключения в конструкторах и деструкторах

```
class Vector {  
public:  
    class Size{};  
    enum {max = 32000};  
    Vector(int n)  
    { if(n<0 || n > max) throw Size(); }  
};  
  
try {  
    Vector *p = new Vector(i);  
    ...  
}  
catch(Vector::Size) { //Обработка ошибки размера  
                        вектора }
```

Исключения в конструкторах и деструкторах

Если в конструкторе генерируется исключение то автоматически вызываются деструкторы для полностью созданных в этом блоке объектов. Например, если исключение было вызвано при создании массива объектов, то деструкторы будут вызваны только для успешно созданных элементов. Если память выделяется динамически с помощью операции `new` и в конструкторе возникает исключение, память из-под объекта корректно освобождается.

- `bad_alloc`
- `bad_cast`
- `bad_typeid`
- `bad_exception`

Файловые потоки:

- `ifstream` — входной файловый поток
- `ofstream` — выходной файловый поток
- `fstream` — двунаправленный

```
char buf[100];  
ifstream fff("file.txt", ios::in | ios::nocreate);  
if(! fff)  
    cout << "err";  
while(! fff.eof())  
    fff.getline(buf, 100);
```

Строковые потоки:

- `istringstream` — входные строковые потоки;
- `ostringstream` — выходные строковые потоки;
- `stringstream` — двунаправленные строковые потоки.

```
#include <sstream>
ostream os;
time_t t;
time(&t);
os << "␣time:␣" << ctime(&t);
```

Потоки и типы определенные пользователем

```
friend ostream& operator << (ostream& out, MyClass&
    C)
{ return out << "x=" << C.x << "y" << C.y; }

friend istream& operator >> (istream& in, MyClass&
    C)
{ cout << "Enter x:"; in >> C.x;
  cout << "Enter y:"; in >> C.y;
  return in;
}
```

Класс string входит в стандартную библиотеку C++.

```
string s1("Text");  
string s2;
```

```
char *str = "Txt";  
string s3(str);
```

Функции:

```
s.at(1);  
s1.append(s2); // +  
s1.append(s2, 3, 5);  
s1.insert(1, str2, 3, 5);  
s1.replace(1, 2, str2, 5);  
s1.swap(s2);  
s1.c_str(); //возвращает const char*  
s1.find(s2); //возвращает позицию  
s1.compare(1, 2, s2, 1, 2); //аналог strstr
```

Последовательные контейнеры:

- `vector` — структура, эффективно реализующая произвольный доступ к элементам, а также добавление в конец и удаление из конца;
- `deque` — структура, эффективно реализующая произвольный доступ к элементам, а также добавление в оба конца и удаление из обоих концов;
- `list` — список, эффективно реализует вставку и удаление элементов в произвольное место, но не имеет произвольного доступа к своим элементам.

```
vector<int> v;  
v.push_back(5);  
v.push_back(6);  
v.push_back(7);  
for( vector<int>::iterator i = v.begin(); i != v.  
    end(); i++)  
{ cout << *i << " "; }
```

Функции:

```
insert() //вставка в произвольное место  
erase() //удаление из произвольного места  
at, [] //произвольный доступ к элементу
```

Ассоциативные контейнеры:

- map — словарь
- multimap — словарь с дубликатами

```
map<int , int> maps;  
maps[100] = 1;  
maps[200] = 3;
```
