Программирование и алгоритмизация

Санкт-Петербургский государственный политехнический университет

14 апреля 2015

Встроенные статические объекты

```
class A {
  A a; //ошибка использования неопределенного
       класса A
};

class A {
  static A a; //OK
};
```

Альтернатива значениям параметров по умолчанию

Преимущество использования статических полей заключается в том, что такое значение можно заменить в процессе работы программы.

```
//Date.h
class Date {
int day, month, year;
static Date default; //дата запуска приложения

public:

Date() {                    }; //FIXME
};
```

Альтернатива значениям параметров по умолчанию

```
#include <ctime>
#include "Date.h"
Date Date::default = Init();
Date Init()
{
   time_t cur;
   time(&cur);
   tm *t = localtime(&cur);
   return Date(t->tm_day, t->tm_month+1, t->tm_year)
       ;

}
```

Целочисленные статические константы

```
class A
{
  int a1;
  const int a2;
  static int a3;
  static const int a4 = 60;
};
```

Статические методы

Статические методы являются фактически глобальной функцией, область видимости которой ограничена именем класса. При наличии прав доступа можно просто вызвать из функции: имя_класса::имя_функции();
При необходимости можно вызвать как обычный метод у объекта.

Статические методы

- в таких функциях указатель this не существует;
- обратиться к нестатическим данным класса из этой функции невозможно;
- статическаая функция не может быть virtual.

Статические методы и защищенные поля

```
//X.h
class X
{
  static int count;
public:
  X() { count++; }
  static int GetCount() { return count; }
};
```

Статические методы и защищенные поля

```
#include "X.h"

int X::count; //???
int main()
{
   cout << X::count; //???
   cout << X::GetCount(); //???
   X x1;
   x1.GetCount(); //???
}
```

Обработка исключительных ситуаций

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 исключительных ситуаций

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.

Обработка исключительных ситуаций

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 дейстия;
- передается объект и управление обработчику имеющему совместимый тип с этим объектом.

Обработка исключительных ситуаций

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 дейстия;
- передается объект и управление обработчику имеющему совместимый тип с этим объектом.

Обработка исключительных ситуаций

Обработчик считается найденым, если:

- тип объекта в обработчеке 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* .

Обработка исключительных ситуаций

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
}
```

Обработка исключительных ситуаций

При переопределении виртуальной функции можно задавать список исключений такой же или более ограниченный чем в базовом классе.
Если функция вызывает не описанное исключение, выывается 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.

Стандартные исключения

- bad_alloc
- bad_cast
- bad_typeid
- bad_exception

Файловые потоки:

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

Файловые потоки

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

Стандартная библиотека C++

Строковые потоки:

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

Файловые потоки

```
#include <sstream>
ostringstream 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;
}
```