Технология ООП

Санкт-Петербургский государственный политехнический университет

27 сентября 2011

Шаблоны классов

```
class Queue {
public:
    Queue();
    ~Queue();

    int& remove();
    void add( const int & );
    bool is_empty();
    bool is_full();
private:
};
Queue qi;
```

Шаблоны классов

```
template <class Type>
class Queue {
public:
    Queue();
    ~Queue();

    Type& remove();
    void add( const Type & );
    bool is_empty();
    bool is_full();
private:
};
Queue<int> qi;
```

Шаблоны классов

```
typedef double Type;
template <class Type>
class QueueItem {
public:
private:
    Type item;
    QueueItem *next;
};
```

Шаблоны классов

```
template<class T = char> class String;
String<> *p;
```

Шаблоны классов

```
template<class List>
class List
{
public:
void print();
};
template <class Data> void List<Data>::print();
```

Шаблоны классов

```
template<class List>
class List
{
public:
void print();
};
template <class Data> void List<Data>::print();
```

Шаблоны классов

```
template< template<class U> class V> class C {
V<int> y;
V<int*> y1;
};
```

Шаблоны классов

```
#include <iostream>
template <int N>
struct Factorial{
enum { val = Factorial<N-1>::val * N };
};
template<>struct Factorial<0>{
enum { val = 1 };
};
int main(){
std::cout << Factorial<4>::val << "\n";} //на этапе
    компиляции
```

Шаблоны классов

```
template<class T> void f() { T::x * p; ... }
template<class T> void f() { typename T::x * p; ...
    }
```

Не стоит злоупотреблять шаблонами

```
../global.h:145: error: in passing argument 3 of `
    int vstd::findPos(const std::v
ector<T1, std::allocator<_CharT> >&, const T2&,
    Func&) [with T1 = std::pair<cons
t CGHeroInstance*, CPath*>, T2 = const
    CArmedInstance*, Func = boost::_bi::bind_
t<bool, bool (*)(const std::pair<const
    CGHeroInstance*, CPath*>&, const CGHeroIn
stance* const std::pair<const CGHeroInstance*,
    CPath*>::*, const CArmedInstance*
 const&), boost::_bi::list3<boost::arg<1> (*)(),
     boost::_bi::value<const CGHerol
nstance*std::pair<const CGHeroInstance*, CPath
    *>::*>, boost::arg<2> (*)()> >]'
```

Спецификаторы класса памяти

- auto
- register
- static
- extern

inline-функции

```
inline double Sqr(double x) return x*x;
```

Спецификатор volatile

```
volatile int i;
```

Преобразования типов

Операция const_cast.

```
void print(int *p) { cout << *p; }
const int *p;
print(p); //ошибка
int *j = const_cast<int*>(p);
```

mutable

```
class mutable_test
{
  int       data;
  mutable sync  sync_obj;
public:
  void set_data (int i)
  {
      sync_obj.lock ();
      data = i;
      sync_obj.unlock ();
  }
  int get_data () const
  {
      sync_obj.lock ();
      int i = data;
      sync_obj.unlock ();
      return i;
  }
};
```

Преобразование типов

Операция dynamic_cast. Преобразования бывают двух типов:

- повышающее – приведение производного класса к базовому
- понижающее — из базового класса в производный
- перекрестное — приведение между производными типами

dynamic_cast

Повышающее преобразование.

```
class B{ ... };
class C : public B{ ... };
C *c = new C;
B *b = dynamic_cast<B*>(c); //Эквивалентно B *b = c
    ;
```

dynamic_cast

Понижающее преобразование. Применяется когда компилятор не может проверить правильность приведения. Для использования данного типа преобразований необходимо включить механизм RTTI.

```
class B{ public: virtual void f1() {} };
class C : public B{ public: void f2() {} };
C *c = new C;
B *b = new B;
C *temp = dynamic_cast<C*>(b);
if(temp)
temp->f2();
```

dynamic_cast

Перекрестное преобразование.

```
class B{ public: virtual void f1() {} };
class C : public B{ public: void f2() {} };
class D : public B{ ... };
D *d = new D;
C *temp = dynamic_cast<C*>(d);
if(temp)
temp->f2();
```

dynamic_cast

Для доступа к RTTI введена операция typeid и класс type_info.

```
class B { ... };
class C : public B { ... };
B *p = new C;
if(typeid(*p) == typeid(C))
{
dynamic_cast<C*>(p)->f2();
cout << typeid(*p).name();
}
```

static_cast

Выполняется на этапе компиляции над:

- целыми типами
- целыми и вещественными типами
- целыми и перечисляемыми типами
- указателями и ссылками одной иерархии

reinterpret_cast

Применяется для преобразования не связных между собой типов, например указателей в целые и наоборот.

```
char *p = reinterpret_cast <char*>(malloc(10));
```