Язык C++

Мещерин Илья

Лекция 9

Шаблоны

6.1) Объявление шаблонов

```
//template<class T>
template<typename T>
void swap(T &a, T &b){}
```

Это называется *шаблон функции*. Первая и вторая строчки эквивалентны. T используется как название типа.

```
typename<typename T>
class C {};
```

Это называется *шаблон класса*.

```
int a, b;
swap<int>(a, b);
swap(a, b);
```

При вызове шаблонной функции компилятор умеет определять тип T без явного указания на это.

```
C<int> c;
```

При объявлении объекта класса нужно указывать шаблонные параметры.
Шаблоны еще одно проявление полиморфизма. Шаблоны - статический полиморфизм, виртуальные функции - динамический полиморфизм. Статический, т.е. разрешается на этапе компиляции, динамический - на этапе выполнения. Компилятор на этапе компиляции генерирует код функции *swap()*, подставляя вместо T тип, например, *int*, и вызов ее подставляет в нужное место.
Способы реализации полиморфизма в C++ – перегрузка функций(статический полиморфизм), виртуальные функции, шаблоны.

6.2) Специализации шаблонов

```
template<class T>
void f(T &x);

void f(int &x);
```

Предпочтение при вызове функции отдается более частному случаю (более специализированному).

```
1 template<class T, class Z>
2 void f(T &x, Z &y) { cout << 1 << "\n"; }
3
4 template<class T>
5 void f(int &x, T&y) { cout << 2 << "\n"; }
6
7 void f(int &x, int &y) { cout << 3 << "\n"; }
```

```
 8
 9  int main(){
10      int x;
11      double y;                          stdout:
12      f(x, x);                                  3
13      f(x, y);                                  2
14      f(y, x);                                  1
15      f(y, y);                                  1
16  }
```

Компилятор в первую очередь рассматривает точное совпадение типов, не рассматривая приведение типов, а после этого предпочтет ту, которая более специализирована.

```
template<class T>
void f(T &x, double &y) { cout << 4 << "\n"; }
f(x, y);
```

Если еще добавить четвертую функцию и сделать такой запрос, то будет ошибка компиляции из-за неоднозначности.

6.3) *Typedef*

Введение нового названия для уже существующего типа.

```
typedef set<int, vector<int>> mytype;
```

Компилятор на этапе компиляции будет вместо *mytype* подставлять указанное выражение.

```
template<typename T>
using mytype = set<int, vector<T>>;
mytype<int> d;
```

Шаблон *typedef* - третий вид шаблонов.

6.4) Пример

```
template<class T>
class C{
public:
    typedef T type;
};

template<class T>
int f(){
    typename C<T>::type x;
}
```

Здесь работает следующее правило: пока компилятор не подставил конкретное T, он не понять, когда $C<T>::type$ является названием поля, а когда названием типа. Если возникает такая неоднозначность, то компилятор считает, что $C<T>::type$ является названием поля. Чтобы явно указать, что $C<T>::type$ является названием типа нужно написать *typename*.

6.5) *Remove_const, remove_reference, remove_pointer*

```
template<class T>
struct remove_const{
    typedef T type;
};
```

```
template<class T>
struct remove_const<const T>{
    typedef T type;
};

typename remove_const<T>::type x;
```

В данном случае тип переменной x будет T, но только не константный, если он был таковым.

6.6) Пример

```
template<class T>
void f(T x);
int x;                f(x);   T = int
int &y = x;           f(y);   T = int
const int z = 0;      f(z);   T = int
const int &t = z;     f(t);   T = int

void f(T &x);
int x;                f(x);   T = int,          int&       x
int &y = x;           f(y);   T = int,          int&       x
const int z = 0;      f(z);   T = const int,    const int& x
const int &t = z;     f(t);   T = const int,    const int& x
```

6.7) Non-type template parameters

```
template<class T, int n>
class array{};
array<int, 10> a;
```

Тип *double* использовать нельзя.

6.7$\frac{1}{3}$) Template template parameters

```
template<class S, class W>
struct D{};

template<class S, class W, template<class, class> class T>
void f(T<S, W> &t){}

D<int, double> d;
f(d);
```

Есть три вида шаблонных параметров – параметры шаблонов, являющиеся типами, параметры шаблонов, являющиеся константами, и параметры шаблонов, являющиеся шаблонами.

6.7$\frac{2}{3}$) Значение по умолчанию

Шаблонные параметры, как и параметры функции, допускают значение по умолчанию.

```
template<class T = int>
void f(){}
```