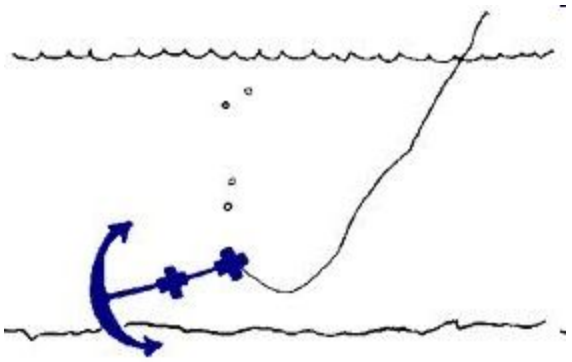
Карпов В.Э.

Объектно-ориентированное программирование

C++. Лекция 8



ШАБЛОНЫ

Предположим, что перед нами стоит задача создания списка элементов некоторого типа. Например, списка целых чисел. Реализация может выглядеть примерно так:

```
class List
{
  int val;
  List *next;
public:
  void add(List *e);
  void add(int n)
  {
    List *t = new List(n);
    add(t);
  }
  void print(void);
  List(int v0) { val = v0; next = NULL; };
};
void List::add(List *e)
{
  List *t;
  for(t=this;t->next;t=t->next);
  t->next = e;
}
void List::print(void)
{
  for(List *t=this;t;t=t->next)
    printf("%d ", t->val);
}
```

Пример работы со списком:

```
List L=1;
L.add(&List(2));
List *pl = new List(3);
L.add(pl);
L.add(4);
L.print();
```

Понятие шаблона

Введено в работе Б.Строуструпа "Parameterized Types for C++", 1988.
Определение шаблона:

<template> <список параметров> объявление

Объявление описывает функцию или класс. Объявление может быть только глобальным.

```
template <class T> class Vector
{
  T* v;
  int sz;
public:
  Vector(int n) { sz = n; v = new T[sz]; };
  void sort(void);
  T& operator[](int i) { return v[i]; }
};
Vector <int> v1(20);
Vector <Complex> v2(10);
typedef Vector <Complex> CVector;
```

Работа с именем шаблонного класса ведется так же, как и просто с именем класса.

```
class SVector: public Vector<Complex> {…}
```

Шаблоны функций-членов

template <class T> T&Vector<T>::operator[](int i) {...};

template <class T> void Vector<T>::sort(void) {...};

При использовании шаблонных функций на компилятор ложится работа по определению их реализации на основе анализа их типа.

Vector <Complex> cv(100);

Vector <int> ci(100);

f(cv); //f(vector<Complex>)

f(ci); //f(vector<int>)

Шаблоны функций

```
template <class T> T max(T a, T b) {return
    a>b?a:b;};
int a, b;
char c, d;
int m1 = max(a,b); //max(int, int)
char m2 = max(c,d); //max(char, char)
int m3 = max(a,c); // - ошибка: нельзя
    сгенерировать max(int, char)
```

Ограничения

Каждый параметр шаблона, заданный в списке параметров, должен быть использован в типах параметров шаблона функции.

```
template <class T> T* create(); // Ошибка
template <class T> void f()
{
  T a;
…
}; // Ошибка
template <class T> class creator
{
  static T* create();
}
int *creator<int>::create();
```

Пример

```
struct S
{   int a, b; };

template <class T> T* create()
{       T *a;
        a = new T;
        return a;
}
template <class T> class Ccreator
{       public:
          static T* create();
};

int *Ccreator<int>::create()
{  int *n;
    n = new int;
   *n = 123;
   return n;
}
```

```
void main(void)
{
  int *t;
  S *s;
  t = create<int>();
  s = create<S>();
  t = Ccreator<int>::create();
}
```

Примеры

```
template <class T> class TList
{public:
  T val;
  TList *next;
  void add(TList *e);
  void add(T n)
  { TList *t = new TList(n);  add(t);   }
  void print(void);
  TList(T v0) { val = v0; next = NULL; };
};
template <class T> void TList<T>::add(TList *e)
{
  TList *t;
  for(t=this;t->next;t=t->next);
  t->next = e;
}
template <class T> void TList<T>::print(void)
{
  for(TList *t=this;t;t=t->next)
    printf("%d ", t->val);
}
```

```
template <class T> void print(T e) {
printf("%d ", e.val); }
// Вплоть до момента вызова этой
функции компилятор ничего не знает о
// типе аргумента
template <class T> void print()
{
  T x;
  printf("%d ", 1);
}

class C
{
  int n, n2;
public:
  C(int k=0) { n=k; n2=n-1;}
};
```

Продолжение

```
void main(void)
{
  TList<int> tl=1;
  tl.add(&TList<int>(2));
  TList<int> *tpl = new TList<int>(3);
  tl.add(tpl);
  tl.add(4);
  tl.print();
  typedef TList<char> CList;
  CList cl=1;
  cl.add(&CList(2));
  CList *cpl = new CList(3);
  cl.add(cpl);
  cl.add(4);
  cl.print();
  TList<C> ctpl = C(31);
  ctpl.add(32);
  ctpl.print();
  print(ctpl);
  print(); //Ошибка: не найден образец для print<T>()
  print<int>(); //А вот так можно
}
```

Множество параметров

Параметров шаблонов может быть несколько:

```
template <class A, class B> void print(A e, B x) {
    printf("%d %d", e.val, x); }
template <class T1, class T2> class L12
{
  T1 t1;
  T2 t2;
};
...
print(ctpl, 1);
L12<int, int> s;
```