Лекция 8

ШАБЛОНЫ ФУНКЦИЙ.

ШАБЛОНЫ КЛАССОВ.

Шаблоны функций (template)

Рассмотрим некоторую функцию, например, **max(x, y)**, аргументы которой могут быть любого типа. Один путь решения этой задачи – перегрузка функций:

```
int max(int x, int y) {return (x > y) ? x : y;}
long max(long x, long y) {return (x > y) ? x : y;}
```

Тогда для каждого типа нужно описать свою функцию.

Второй путь – использование макросов:

```
#define max(x, y) ((x>y)?x:y)
```

Но тогда нет проверки типов, и возможно сравнение несравнимых типов (например, **int** и **struct**). Также замена в исходном тексте макроса выполняется даже тогда, когда это не требуется, например:

```
class My_class {
     // ...
int max(int, int);    //Объявление метода
};                    //при компиляции будет выполняться подстановка
```

Т.е. макросы в этом случае использовать нельзя!

Шаблоны функций

Для решения таких задач используются шаблоны.

Шаблон функции представляет собой обобщенное определение функции, которое конкретизируется в момент вызова для походящего типа. Компилятор создает по шаблону функции конкретного ее представителя (**порожденную функцию**).

Синтаксис объявления шаблона:

```
template <class T1|T1 идент1, class T2|T2 идент2, …,
          class Tn|Tn идентn>
 возвр_тип имя_функции (параметры){
       /*тело функции*/
  }
```

За ключевым словом **template** следуют один или несколько параметров, заключенных в угловые скобки, разделенные между собой запятыми. Каждый параметр является

- либо ключевым словом, за которым следует имя типа;
- либо именем типа, за которым следует идентификатор.

Вместо ключевого слова **class** может использоваться **typename**.

Шаблоны функций

Пример

```
template <typename T> // определение функции-шаблона Sqr
T Sqr(T x)
     {return x*x;}

int main(){
    int i;
    cout<< "Введите i"<<endl;
    cin>>i;
    cout<< "i*i="<<Sqr(i)<<endl;   // генерируется Sqr() для int

   double d;
   cout<< "Введите d"<<endl;
   cin>>d;
   cout<< "d*d="<<Sqr(d)<<endl;   // генерируется Sqr() для double
}
```

Шаблоны функций

Пример

```
template <class T> // определение функции-шаблона max
T mymax(T x, T y)
      { return (x > y) ? x : y; }

int main() {
int i1=3, i2=5;
float f1=3.3, f2=5.5;
long l1=76543L, l2=345678L;
cout <<"max(i1,i2)="<< mymax(i1,i2)  // генерируется max() для int
     <<" max(f1,f2)="<< mymax(f1,f2) // генерируется max() для float
     <<" max(l1,l2)="<< mymax(l1,l2) // генерируется max() для long
      << endl;
}
```

```
max(i1,i2)=5 max(f1,f2)=5.5 max(l1,l2)=345678
```

Шаблоны функций

В рассмотренном выше примере функция-шаблон **max()** применима к любым типам, для которых определена операция ">".

Можно добавить исключение для шаблона с конкретным типом, явно определив функцию, например, определим функцию **max** для строк:

```
char * mymax(char *s, char *t) {
        return (strcmp(s, t) > 0) ? s : t;
}
```

Теперь обращение к функции **max(s, t)** для строковых аргументов **не будет генерировать** новую функцию, а вызовет вышеописанную функцию:

```
int main() {
//…
char s1[]="Магнитогорск", s2[]="Москва";
cout<<"max(s1, s2)="<<mymax(s1, s2); // вызывается max() для char*
}
```

```
max(s1, s2)= Москва_
```

Шаблоны функций

Пример: обмен значениями двух элементов массива

```
template <class T>
T* Swap(T* mas, int ind1, int ind2) {
    T temp=mas[ind1];
      mas[ind1]=mas[ind2];
      mas[ind2]=temp;
  return mas;
}

int main(){
char m_char[]="abcdf";
    cout<<"string=>"
            << Swap(m_char, 0, 3)<<endl;
int x[]={0, 1, 2, 3, 4, 5};
int *y=Swap(x, 0, 3);
 for(int i=0; i<6; i++)
        cout<<"massiv=>"<<y[i]<<endl;
}
```

```
string=>dbcaf
massiv=>3
massiv=>1
massiv=>2
massiv=>0
massiv=>4
```

Шаблоны функций

При использовании функций-шаблонов необходимо внимательно следить за типом передаваемых аргументов, поскольку для функций-шаблонов **предопределенные преобразования типов *не* выполняются**, например:

```
template <class T>                    // определение функции-шаблона
T mymax(T x, T y) {
  return (x > y) ? x : y;
}
int main() {
 int i=5; char c=3;
 int x1, x2, x3, x4;
     x1=mymax(i, i); // все в порядке
     x2=mymax(c, c); // также все в порядке
 x3=mymax(i, c);     // не работает, поскольку аргументы разного типа
 x4=mymax(c, i);     // не работает, поскольку аргументы разного типа

  cout<<"x1="<<x1<< "x2="<<x2<<"x3="<<x3<<"x4="<<x4;
}
```

Шаблоны функций

Чтобы допустить предопределенные преобразования типов при обращении к функции-шаблону, достаточно явно определить ее прототип в шаблоне:

```
template <class T>  // определение буквы шаблона
T mymax(T x, T y)  // определение функции-шаблона
   {  return (x > y) ? x : y; }
int mymax(int, int);  // определение прототипа max()

int main() {
int i=5;  char c=3;   int x1, x2, x3, x4;
 x1=mymax(i, i);  // все в порядке
 x2=mymax(c, c);  // также все в порядке
 x3=mymax(i, c);  // теперь уже работает
 x4=mymax(c, i);  // теперь уже работает
 cout<<"x1="<< x1 <<"x2="<< x2
      <<"x3="<< x3 <<"x4="<< x4 << endl;
}
```

Ограничением на использование функций-шаблонов является то, что для них **нельзя** указывать умалчиваемые значения аргументов.

Шаблоны функций, использующие несколько типов

Очень часто в шаблоне функции требуется *указать несколько типов*. Например, опишем шаблон для функции **show_array**, которая выводит элементы массива. Шаблон использует тип **T** для определения типа массива и тип **T1** для указания типа параметра **count**:

```
template<class T, class T1>
void show_array( T *array, T1 count)
{ T1 index;
for (index = 0; index < count; index++)
       cout << array[index] <<" ";
 cout << endl;
}
void show_array(int *, int); // прототипы
void show_array(float *, unsigned);
```

```
100 200 300 400 500
11.01 22.02 33.02
```

```
int main() {
 int pages[] = { 100, 200, 300, 400, 500 };
 float prices[] = { 11.01,  22.02,  33.02 };
 show_array(pages, 5); // генерируются порожденные функции
 show_array(prices, 3);
}
```

Пример шаблона функции, использующий несколько типов

```
template <class T1, class T2> // описание шаблона
void Show(T1 x, T2 y) {
 cout<<setw(8)<<x<<endl;
  cout<<setw(8)<<y<<endl;
}

int main(){
  int i;  double d;
cout<<"Введите i"<<endl;
     cin>>i;
cout<<"Введите d"<<endl;
     cin>>d;
Show(i, d);   // генерируется и вызывается Show (int, double)
Show(d, i);   // генерируется и вызывается Show (double, int)
}
```

Шаблон класса

Шаблон класса (**class template**), называемый также **обобщением классов** (**generic class**) или генератором классов (**class generator**), задает образец для определений классов.

Рассмотрим шаблон класса **Array**, который позволяет определить классы массивов с любым типом элементов:

```
template <class T> // определение шаблона классов Array
class Array {
   T *m; // тип массива задается шаблоном
   int size;
 public:
      Array(int n);
      ~Array();
 void print (const char*);   // описание метода печати массива
 // метод записи x в i-ый элемент массива
void set(int i, T x) { *(m+i) = x; }
// перегрузка операции индексации
// для обращения к элементу массива
 T & operator [] (int i) { return m[i]; }
};
```

Шаблон класса

```
template <class T>
Array<T>::Array(int n){
      size = n;
      m = new T[size];
    }
template <class T>
Array<T>::~Array() {delete [] m;}

// определение метода печати массива
 template <class T> // шаблона классов
void Array<T>::print(const char *s)
{ cout << s << ":";
  for (int i=0;  i<size; cout<<m[i++]<<" ");
  cout << "\n";
}
```

Шаблон класса

Теперь можно определять объекты различных классов **Array,** задавая в качестве параметра шаблона конкретный тип элементов массива:

```
int main()
{
 Array<int> mi(5);  // объект класса Array для int
 Array<float> mf(3);  // объект класса Array для float
int i;
 for (i=0; i<5; i++)
     mi.set(i, i);    // заполнение массива int
         mi.print("mi");    // печать массива int
 for (i=0; i<3; i++)
     mf.set(i, i );    // заполнение массива  float
           mf.print("mf");  // печать массива float
}
```

```
mi:0 1 2 3 4
mf:0 1 2
```

Шаблон класса

В общем случае определение шаблона классов имеет вид:

```
template <список_аргументов>
class имя_класса {/*определение класса*/};
```

Типы могут быть как стандартными, так и определенными пользователем. Для их описания в списке аргументов используется ключевое слово **class**. В простейшем случае параметр один.

```
template <class T>
class My_class {     //…
};
```

Здесь **T** является параметром-типом. Имя параметра может быть любым, но принято начинать его с префикса **T**.

Методы шаблона класса ***автоматически*** становятся **шаблонами функций**. Если метод описывается вне шаблона, его заголовок должен иметь следующие элементы:

```
template <список_аргументов>
<возвр_тип> имя_класса :: имя_функции (список_параметров)
                          {\*тело функции*\}
```

Шаблон класса

Объекты конкретного класса задаются следующим образом:

```
имя_класса <аргументы_шаблона> список_объектов;
```

Аргументы шаблона классов могут быть двух видов: типовые и не типовые.

Типовым аргументам предшествует ключевое слово **class**, они, как правило, обозначаются буквой и представляют параметр типа, т.е. изменяемые типы данных.

Не типовые аргументы шаблона классов аналогичны параметрам конструктора класса, для них можно задавать умалчиваемые значения параметров, определяемые константными выражениями. При этом каждый генерируемый класс будет получать собственную копию статических компонент.

В следующем примере тип элементов массива и размер массива передаются в качестве параметров шаблона классов:

Шаблон класса

```
template <class T, int n=1> // определение шаблона
class Array{
      T *m; // тип массива задается шаблоном
      int size;
public:
Array() {
      size = n; // размер массива задается параметром шаблона
      m = new T[size]; // тип памяти задается шаблоном
      }
~Array();
T & operator[](int i) {return m[i];} // перегрузка опер. индексации
void print(const char*);
};

template <class T, int n> Array<T, n>::~Array(){
            delete []m;}
```

Шаблон класса

```
template <class T, int n> //определение
void Array<T, n>:: print(const char *s){
  cout << s << ":";
  for (int i=0; i<size; cout << m[i++] << " ");
  cout << "\n";
}

int main(){
Array<int, 3> mi; // определение класса Array для int
Array<float, 5> mf; // определение класса Array для float
int i;
for (i=0; i<3; i++) mi[i] = i; //заполнение
mi.print("mi");              //печать
for (i=0; i<5; i++) mf[i] = i;  // заполнение
mf.print("mf");               // печать
}
```

Шаблон класса

- Описание параметров шаблона в заголовке функции **должно** соответствовать шаблону класса.
- Локальные классы **не могут** иметь шаблоны в качестве своих элементов.
- Шаблоны методов **не могут** быть виртуальными.
- Шаблоны классов **могут** содержать статические элементы, дружественные функции и классы.
- Шаблоны **могут** быть производными как от шаблонов, так и от обычных классов, а также являться базовыми и для шаблонов, и для обычных классов.
- Внутри шаблона **нельзя** определять **friend**-шаблоны. Если у шаблона несколько параметров, они перечисляются через запятую.

Параметрам шаблонного класса можно присваивать значения по умолчанию.

Достоинства и недостатки шаблонов

Шаблоны представляют собой мощное и эффективное средство обращения с различными типами данных, которое можно назвать **параметрическим полиморфизмом**, *обеспечивают безопасное использование типов*, в отличие от макросов препроцессора, и являются вместе с шаблонами функций средством реализации идей обобщенного программирования и метапрограммирования. Однако следует иметь в виду, что эти средства предназначены для грамотного использования и требуют знания многих тонкостей.

Программа, использующая шаблоны, содержит код для каждого *порожденного типа*, что может увеличить размер исполняемого файла. Кроме того, с одними *типами данных* шаблоны могут работать не так эффективно, как с другими. В этом случае имеет смысл использовать *специализацию* шаблона. Стандартная библиотека C++ предоставляет большой набор шаблонов для различных способов организации хранения и обработки данных.