10.Функции- и классы-шаблоны

10.1 Функции-шаблоны (родовые функции)

Если функции выполняют одинаковые действия, но над данными *разных типов*, то можно определить **одну функцию-шаблон**.

Определение имеет формат

template <class X[, class Y,…]>определение_функции

Причем вместо типа какого-либо аргумента (или всех) или возвращаемого значения может стоять X, Y,…, где X, Y,… - идентификаторы для обозначения произвольного типа данных.

Задача: отсортировать массив целых чисел и строк методом пузырька

Вообще говоря, потребовалось бы создать 2 функции для сортировки каждого массива и 2 для обмена двух значений, т.е. 4 функции.
Вместо этого зададим 2 функции-шаблона.
Определим дополнительно еще 2 функции-шаблона:

– для нахождения max элемента в массиве данных и

- вывода на экран двух значений данных разного типа.

include "string.hpp“
#include “complex.hpp”

```
// Функции-шаблоны
 template <class T> inline void Swap(T &a,T &b)
{ T c;
  c = a, a = b, b = c;
 }
 template <class T> T Max( T *a, int n)
 { int i; T max = a[0];
  for( i = 1; i<n; i++)
  if (a[i] > max) max = a[i];
  return max;
  }
```

```
template <class T> void Bul_bul( T *a, int n)
// улучшенный «пузырек» с флагом f
{ int i, j, f;
 for( i = 0, f = 1; i<n - 1 && f; i++)
  for( j = 0, f = 0; j<n - i - 1; j++)
  if (a[j] > a[j+1] ) { Swap(a[j], a[j+1]); f =1;}
}
template <class X,class Y> void out2(X  &a, Y  &b)
  { cout<<a<<' '<<b;
  }
```

Как же будет работать компилятор?

При вызове функций-шаблонов в зависимости от ***типов фактических* аргументов** компилятор создает **различные версии** каждой из этих функций. Говорят - компилятор создает *порожденную функцию*.

Процесс генерации порожденной функции называют созданием экземпляра функции. Таким образом, *порожденная функция – это конкретный экземпляр функции-шаблона*.

Примеры

```
void main()
{ String s[5] = {"Петров",«Иванова",«Сидорова",
                 “Иванов", «Петрова"};
  int a[7] = {5, 3, 9, 6,111, 7, 4}, i;
 Complex b[4] = {Complex(4,2), Complex(-3, 4), -2};
  cout<<"\n Max среди строк "<<Max(s, 5);
                          // Генерируется экземпляр
                          //  String Max(String*, int)
  cout << "\n Max в массиве a “ << Max(a, 7);
                          //  Генерируется экземпляр
                          //    int Max(int *, int)
```

Сидорова

111

```
Bul_bul(s, 5); // Генерируется экземпляр
                //  void Bul_bul(String *, int );
 cout<<"\nОтсортированные  строки ";
 for( i = 0; i<5; i++)
  out2(s[i], ' ');  // генерируется экземпляр
                    // void out2(String , char );
Bul_bul(a, 7); // генерируется экземпляр
                // void Bul_bul(int *, int);
 cout<<"\nОтсортированные числа\n";
 for( i = 0; i<7; i++)
 out2( a[i], ' ');    // генерируется экземпляр
                      //  void out2( int , char );
```

```
 cout<<"\nКомплексные числа\n";
  for( i = 0; i<4; i++)
  out2(b[i], "  ");
                        // генерируется экземпляр
                      // void out2(Complex, char *)
  out2(endl, "******* the End *******");
                       // генерируется экземпляр
                       // void out2(char, char *)
}
```

Итого, порождены **8** функций!

И это ещё не всё

При порождении экземпляров функции-шаблона **Bul_bul(...)**

компилятор сгенерирует также 2 функции Swap –

void Swap (String &, String &)

и

void Swap(int &, int &).

В итоге компилятор сгенерирует **10** функций (вместо четырех, заданных программистом)!

Программа будет работать при выполнении следующих условий:

- в классе **String** должна быть **перегружена** операция сравнения на **>** ;
- в классах **String** и **Complex** должна быть **перегружена** операция потокового вывода **<<**;
- в классе **String**, кроме того, должна быть **перегружена** операция **=**.

10.2 Классы-шаблоны

Так же, как и функции – шаблоны, могут быть определены и **классы-шаблоны**.

Определение. **Класс-шаблон** – это класс, в котором одни и те же член-данные могут быть разных типов, а действия над ними - одинаковые.

Формат определения класса-шаблона

```
template   <class X[, class Y,...]>  class
   имя_класса
     { X определение ч-данных;
       Y определение ч-данных;
        .......
      };
```

Здесь X и Y - условное обозначение типов данных.

Использование такого класса в функциях следующее:

имя_класса **<**список_конкретных_типов**>** имя_объекта[(аргументы конструктора)];

При компиляции ***для каждого списка типов данных, указанного в < >,*** будет создан свой ***экземпляр класса*** путем замены условных типов **X**, **Y**,.. на указанные в **< >**.

Пример класса-шаблона Stack

В стек кладутся целые и комплексные числа, строки.

```
# include "string.hpp"
# include "Complex.hpp”
template <class T> class Stack
  { T *a; int max, n;
            // max – макс размер, n – указатель
    public:
    Stack( int k = 50)
     { a = new T [max = k];
       n = 0;
     }
```

```
~Stack() { delete [] a; }  // деструктор
Stack( Stack <T> &);   // конструктор
                       // копирования
Stack <T> & operator = (Stack <T> &);
void Push(T b)
                // положить в стек элемент b
{  if(Full())
       throw "Стек переполнен!";
    a[ n++ ] = b;
}
```

Функции-предикаты

```
bool Empty() // стек пуст?
   { if(n == 0)
     return true;
     return false;
   }
  bool Full();  //  Стек заполнен?
```

```
T Pop()    // удалить элемент из стека
      { if ( ! Empty())
        return a[ --n];
        throw “Стек пуст”;
    }
T Top() // взять значение с вершины стека
 { if (n)
      return a[n-1];
   throw “Стек пуст”; }
```

Потоковый вывод

```
friend ostream & operator<<(ostream & r,
Stack <T> &b)
 { int i;
   if( b. Empty()){ r << “\nСтек пуст”;
                          return r;}
   for( i = 0; i<b.n; i++) cout << ' ' << b.a[i];
   return r;
   }
```

Конструктор копирования

```
template<class T> Stack<T > :: Stack
(Stack <T> & b)
{ int i;
 a =  new T [max = b.max];
   n = b.n;
  for( i = 0; i<n; i++)
   a[i] = b.a[i];
}
```

Full()

```
template <class T>  bool Stack <T> :: Full()
{ f( n == max) return true; // да, заполнен
   return false;              // нет, не заполнен
  }
```

operator =

```
template <class T> Stack<T> & Stack <T> ::
   operator = (Stack <T> &b)
{
 delete [] a;
 a = new T [max = b.max]; n = b.n;
 for( int i = 0; i<n; i++)
  a[i] = b.a[i];
  return *this;
}
```

Примеры использования

```
void main()                                  ?  max = 50 по умолчанию
{ Stack <Complex> c(5),cc;// создается экземпляр стека
                          // для max=5 комплексных чисел
 Stack <String> s(5); // создается экземпляр стека строк
 Stack <int> a(2);    // стек для max=2 целых чисел
 Stack <float> b(3);  // стек вещественных чисел
 String b4("bbbb");   // строка “bbbb”
 int i;
 c.Push(Complex(2,3)); // компл. число 2+3*i => в стек
    // комп. чисел
 c.Push(3.5);  // комп. число 3.5+3.5*i => в стек комп.
    // чисел, сначала работает Complex(float)
s.Push(b4); // строка “bbbb” => в стек строк
s. Push(“ddd”);    // строка “ddd” => в стек строк
```

Сначала работает конструктор String(char *), затем функция Push

```
a.Push(77);          // число 77 => в стек целых чисел
a.Push(9);           // число 9 => в стек целых чисел
 a.Push(1185);       // Исключение «Стек переполнен!”
 cout<<"\n Стек комплексных чисел: ";
 cout<<c; /* работают 2 перегруженные операции
    потокового вывода <<
                для классов Stack и Complex */
 cout<<"\n Стек строк: ";
 while(! s. Empty())  // пока не пуст
 cout << s.Pop();
// работает перегруженная операция << в классе String
// Стек строк опустел
 cc = c; /* перегруж операция = переписывает стек
    комплексных чисел с в стек cc */
 cout<<”\n В вершине стека компл. число “<<cc.Top();
 cout<<"\nСтек вещественных чисел b = "<<b;
                        // Сообщение “Стек пуст”
}
```

KStack::2

3,5 + i*3.5

Замечание

Так как действия над данными в классах шаблонах одинаковые, а типы данных могут быть произвольные, то такие классы еще называют ***контейнерными***.
В настоящее время существует большой набор стандартных классов-шаблонов: стеки, очереди, деревья, динамические массивы и др.