Дополнительные возможности при работе с пользовательскими функциями

Параметры со значениями по умолчанию

Чтобы упростить вызов функции, в ее заголовке можно указать значения параметров по умолчанию. Эти параметры должны быть последними в списке и при вызове функции аргументы для них могут опускаться. Если при вызове аргумент опущен, то должны отсутствовать и все аргументы, стоящие за ним, т.к. задавать значения по умолчанию можно только для последних параметров в списке функции.

Значениями параметров по умолчанию могут быть константы или константные выражения.

Параметры по умолчанию проходят проверку типа во время описания функции и вычисляются при вызове.

Пример функции, определяющей сумму от 2-х до 5-ти переменных:

```
            . . .
int Sum (int a, int b, int c = 0, int d = 0, int e = 0)  {
                    // 0 – умалчиваемые значения
    return (a + b + c + d + e);
}
```

```
void main ()
{
   int  x1=1, x2=1, x3=1, x4=1, x5=1,  y2, y3, y4, y5;
   y2= Sum (x1, x2);            //  y2 = 2
// Работают все умалчиваемые значения;
   y3= Sum (x1, x2, x3);      //  y3 = 3
// - два последних значения;
   y4= Sum (x1, x2, x3, x4);     //  y4 = 4
// - одно последнее значение;
   y5= Sum (x1, x2, x3, x4, x5);   //  y5 = 5
        . . .
}
```

Таким образом:

1. Умалчиваемое значение аргумента функции задается при его объявлении в заголовке функции.
2. В начале списка указывают параметры, значения которых будут передаваться всегда.
3. При обращении пропуск умалчиваемых параметров в списке недопустим, т.е. для получения значения $x1 + x2 + x3 + x5$ вызов функции *Sum* ($x1$, $x2$, $x3$, $x5$); приведет к ошибочному результату.

Правильным будет обращение, например

y = *Sum* ($x1$, $x2$, $x3$, 0, $x5$);

Перегрузка функций

В языке C++ реализована возможность использования одного идентификатора для функций, выполняющих различные действия над различными типами данных, в результате чего можно использовать несколько функций с одним и тем же именем, но с разными списками параметров, как по количеству, так и по типу.

Такие функции называют *перегруженными*, а сам механизм – *перегрузка функций*.

Компилятор определяет, к какой из функций следует обратиться путем сравнения типов аргументов с типами параметров.

Пример перегрузки функций

Пример функции ***Fun*** с двумя параметрами *x*, *y*, работающей в зависимости от типа передаваемых аргументов, следующим образом:

– если тип параметров ***int***, функция ***Fun*** складывает их значения и возвращает сумму;

– если тип параметров ***long***, функция ***Fun*** перемножает их значения и возвращает произведение;

– если тип параметров ***double***, функция ***Fun*** делит их значения и возвращает частное от деления.

```
int Fun (int x, int y) {
    return  x + y;
}

long Fun (long x, long y) {
    return  x * y;
}

double Fun (double x, double y) {
    return  x / y;
}
```

```
void main ()
{
   int a = 2, b = 3, s;
   long i = 3, j = 4, pr;
   double x = 10, y = 3, d;
   s = Fun(a, b);
   pr = Fun(i, j);
   d = Fun(x, y),
   cout  << "\n sum = " << s << " pr = " << pr
       << " del = " << d << endl;
}
```

В результате получим:

sum = 5 pr = 12 del = 3.33333

Понятие шаблона функции

Механизм шаблонов – средство построения обобщенных определений функций, независимых от используемых типов данных. Их использование избавляет от необходимости дублировать код функции для различных типов данных, составляющих их параметры и возвращаемые результаты.

Компилятор по заданному в качестве аргумента типу данных на основе определения шаблона автоматически порождает соответствующие экземпляры функций.

Формат шаблона функции:

template *Список параметров шаблона*

Декларация функции;

Список параметров шаблона определяет набор типов. Каждый тип определяется словом ***class*** и является локальным типом данных в рамках функции. Список параметров не может быть пустым.

Декларация функции – обычное определение (или прототип) функции. В списке ее параметров необходимо упомянуть хотя бы один раз типы параметров из списка шаблона.

***Пример* 1**. Шаблон с единственным параметром

```
template  < class T >
void  fun (T par) {
 код функции fun
}
```

Символ типа ***T*** можно использовать и для возвращаемого значения и для любых других объектов в коде функции.

Функция может иметь любые типы параметров как параметризованные, так и стандартно декларированные.

***Пример* 2.** Шаблон функции с частично параметризованными параметрами:

```
template  < class T >
    void  fun (T par, int x, int y) {
        код функции fun
  }
```

***Пример* 3**. Шаблон может иметь несколько параметризованных параметров с разными символическими идентификаторами:

```
template  < class T2, class T1 >
void fun (T1 par1, T2 par2) {
        код функции fun
}
```

Порядок следования идентификаторов параметров в заголовке функции может отличаться от их декларации в шаблоне.

Пример нахождения максимума для разных типов аргументов:

```
#include <iostream.h>
   template < class T >
   T  fun (T x, T y) {
    return (x > y) ? x : y;
   }
```

```
void main()
{
    int  n = 2, m = 3;
    double  x = 7.5,  y = 2.5;
    cout << "Max (" << n << " , " << m << ") = "
      << fun (n, m) << endl;
    cout << "Max (" << x << " , " << y << ") = "
      << fun (x, y) << endl;
}
```

Результат :

Max (2 , 3) = 3

Max (7.5 , 2.5) = 7.5

Перегрузка шаблонов функций

Можно перегружать функции-шаблоны для неподходящих под данный код шаблона данных, т.к. их использование базируется, как и у обычных функций, на распознавании компилятором различий в списках их параметров.

Пример использования преобразования шаблонов и специализированных функций:

```
        . . .
template < class T >
T  max (T x, T y) {
 return (x > y) ? x : y;
}
char* max (char* x, char* y) {
 return ( strcmp (x, y) > 0) ? x : y;
}
```

```
void main()
{
   int a = 7, b = 5;
   char c1 = 'a', c2 = 'z';
   char s1[] = "one", s2[] = "two";
    cout << "Max( " << a << " , " << b << ") = "
         << max(a, b) << endl;
    cout << "Max( " << c1 << " , " << c2 << ") = "
         << max(c1, c2) << endl;
    cout << "Max( " << s1 << " , " << s2 << ") = "
         << max(s1, s2) << endl;
}
```

Результат :

Max (7 , 5) = 5

Max (a , z) = z

Max (one , two) = two