Функции

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

16 декабря 2016 г.

Версия: 0.11

Аннотация

Функции: создание, вызов по значению и возврат значения. Локальные переменные, аргументы. Рекурсивный вызов функции.

1 Функции

Функции полезны при необходимости выделения общего, часто используемого кода. Например, можно написать функцию, которая считывает количество чисел в текстовом файле. Далее, копируя текст этой функции, её можно переиспользовать, а не писать код каждый раз заново. При этом код функции уже будет "рабочим т.е. ранее использованным и тем самым проверенным.

1.1 Создание функций

Покажем как создать функцию, вычислить и вернуть значение.

Без функции Рассмотрим функцию вычисления абсолютного значения. Для начала рассмотрим в очередной раз как выглядит программа без применения функции, где имеет повторяющийся код.

```
1 #include <stdio.h>
2
3 int main()
4 {
5         int a;
```

```
6           scanf( "%d", &a);//Опустим проверку ошибок.
7           if( a < 0 )//Проверка на отрицательность числа.
8                   a = -a;//Отрицание числа.
9           int b;
10          scanf( "%d"     , &b);
11          if( b < 0 )//Повтор тех же самых действий.
12                  b = -b;
13          printf("sum abs val ~ %d\n", a + b);
14          return 0;
15  }
```

В данной функции два раза выполняется одни и те же вычисления, а именно – вычисляется абсолютная величина. Такой подход имеет ряд недостатков. Во-первых, разрастается код, что приводит к его нагромождению и, соответственно, к ухудшению его читаемости. Во-вторых, каждая копия живет отдельной жизнью и в случае модификации (устранение ошибок, расширение возможностей) одной копии будет необходимо внести изменения и во все остальные, что существенно повышает вероятность появления очередных ошибок (человек плохо умеет копировать действия).

Добавим функцию Функция позволяет обособить часть кода в именной (в данном случае, `my_abs`) блок, которому передается список значений (в данном случае, `k`). Передаются именно значения, а не сами переменные причем в общем случае возможно различных типов (в данном случае, `int`). Модификация программы при применении функции:

```
1   #include <stdio.h>
2
3   //Пишем функцию, до её использования. В общем виде:
4   //возвращаемый тип   имя(список типов аргументов)
5   //{
6   //код функции
7   //}
8
9   //Наша функция будет иметь имя my_abs, как функция
10  //возвращать целое значение, тип единственного аргумента
11  int my_abs(int k)//-- целый. Имя аргумента k.
12  {//Переменной k присвоено значению при вызове.
13          if( k < 0 )//Значимый код пишется один раз.
14                  k = -k;//Можно его тщательно проверить.
15          return k;//Возвращаем значение данной функции.
```

```
16 }
17
18 int main()
19 {
20         int a;
21         scanf( "%d", &a);
22         //Вызываем ранее объявленную и описанную функцию.
23         a = my_abs( a );//Коротко и ясно.
24         int b;
25         scanf( "%d"      , &b);
26         b = my_abs( b );//Ошибиться сложно.
27         printf("sum abs val ~ %d\n", a + b);
28         return 0;
29 }
```

Теперь программа стала лаконичнее и яснее.

Функция создана в строчках 11-16. В строчке 11 описывается по порядку возвращаемый тип (`int`), имя функции (`my_abs`) и список входных аргументов (`(int k)`). Сам код функции (фактически, строчки 13 по 15), описывающий соответствующие действия, называется *телом функции*. В строчке 15 выполняется возврат из функции, т.е. фактически завершение работы функции, и указывается её возвращаемое значение.

В самой программе функция вызывается в строчках 23 и 26. При вызове указывается название функции (`my_abs`) и значения соответствующих аргументов. При первом вызове (строчка 23) передается значение переменной `a`, а при втором (строчка 26) – `b`. После завершения вызова функция возвращает значение (как это было описано в самой первой заметке) и в данном случае оно сохраняется в те же самые переменные.

1.2 Возврат значения

Обычно функции возвращают значение, при этом точка возврата не обязательно в конце самой функции (как в предыдущей программе в строчке 15). При грамотном использовании этим понятием можно повысить лаконичность и читаемость кода.

Больше одного return Функция может содержать больше одного return. При правильном использовании это упрощает код, делает его более лаконичным. В согласии с предыдущим, перепишем рассматриваемую функцию.

```
1 int my_abs2(int k)
2 {
```

```
3           if( k < 0 )//Дополнительный возврат из функции.
4                   return -k;//Можно сразу вернуть значение.
5           return k;//Возврат из функции по умолчанию.
6   }
```

Без return по умолчанию Предыдущая функция содержала возврат по умолчанию. Но, если убедить компилятор (а он бывает настойчив относительно того, что не все ветвления ведущие к завершению функции рассмотрены), то можно и без выхода по умолчанию обойтись.

```
1   int my_abs2(int k)
2   {
3           if( k < 0 )//Дополнительный возврат из функции.
4                   return -k;//Можно сразу вернуть значение.
5           else
6                   return k;//Возврат в противном случае.
7   }
```

Тип возвращаемого значения Он может быть произвольным для конкретной функции, но один и тот же у всех вызовов return. При необходимости и возможности компилятор естественно сделает необходимые преобразование типов.

1.3 Аргументы

При вызове функции ей передается список значений от которого она и должна выполнить вычисления.

Больше одного аргумента Естественно, что функция может содержать больше одного аргумента. Например, перепишем предыдущую программу так, что вместо вычисления абсолютного значения, вычисляется минимум двух значений.

```
1   #include <stdio.h>
2
3   //Функция my_min возвращает целое число, но имеет
4   int my_min(int a, int b)//два аргумента: k и m.
5   {//Переменным a и b присвоено значению при вызове.
6           if( a < b )
7                   return a;//Воспользовались возможностью
8                   //преждевременного возврата из функции.
```

```
9           return b;
10  }
11
12  int main()
13  {
14          int a, b;
15          printf("a?:");
16          scanf( "%d", &a);
17          printf("b?:");
18          scanf( "%d", &b);
19          //Вызываем ранее объявленную и описанную функцию.
20          int r = my_min( a, b );//Коротко и ясно.
21          printf("min (%d , %d) ~ %d\n", a, b, r);
22          return 0;
23  }
```

В описание функции (строчка 4) теперь указаны два аргумента (`(int a, int b)`). В согласии с этим, при вызове функции (строчка 20) указываются/передаются два значения (`a` и `b`). Возвращаемое значение сохраняется в переменную `r`.

Без аргументов Интересно, что функция может и не содержать ни одного аргумента. По сути она константная (но есть тонкости!). Фрагмент самой функции:

```
1   #include <math.h>
2
3   //Функция my_pi возвращает вещественное число, но не
4   double my_pi()//имеет аргументов.
5   {
6           return acos(-1);
7   }
8
9   int main()
10  {
11          double a;
12          printf("a?:");
13          scanf( "%lf", &a);
14          double pi = my_pi();//Возвращаем значение ПИ.
15          while( a < 0 )//Пока угол меньше 0
16                  a = a + 2 * pi;//добавляем 2 пи.
17          while( a >= 2 * pi )//По аналогии.
18                  a = a - 2 * pi;
```

```
19          //угол приведен -- от 0 до 2 пи.
20          printf("reduced angle ~ %f\n", a);
21          return 0;
22  }
```

Важно, что в любой момент можно изменить код, отвечающий за вычисление ПИ. В оставшейся программе не придется больше ничего менять.

Например, в любой момент функцию `mypi` можно заменить на:

```
1  //Другой вариант функции
2  double my_pi()
3  {
4          return M_PI;
5  }
```

Тип аргументов Аргумент(ы) функции могут иметь произвольный тип.

1.4 Обособленность функции

Функция позволяет в некотором смысле обосабливать код отвечающий за функцию. Во-первых, за счет локальности переменных, в частности, аргументов. Во-вторых, за счет достижения большей лаконичности и четкости кода. Последние позволяет функции тестировать (отлаживать) отдельно друг от друга.

Вызов по значению Как и в самой первой программе самой первой заметке уже было отмечено, что при перед вызовом функции сначала вычисляются значение всех аргументов. Таким образом этот этап отделен он вычислений в самой функции. Поэтом их можно разрабатывать, тестировать и отлаживать отдельно друг от друга.

Далее эти значения присваиваются аргументам функции и передается управление функции. При вызове return производится возврат из функции в точку вызова. Значение возвращенное из функции будет использовано в арифметических действия в точке вызова.

Это позволяет отделить вычисления аргументов от самой сути функции и от использования самого возвращенного значения функции.

```
1  #include <stdio.h>
2
3  //Функция my_min возвращает целое число, но имеет
4  int my_min(int a, int b)//два аргумента: k и m.
```

```
5  {//Переменным a и b присвоено значению при вызове.
6          if( a < b )
7                  return a;
8          return b;
9  }
10
11 int main()
12 {
13         int a, b;
14         printf("a?:");
15         scanf( "%d", &a);
16         printf("b?:");
17         scanf( "%d", &b);
18         //Найдем максимум квадрата a и b.
19         //Перед вызовом функции my_min сначала
20         //вычисляться значения аргументов, т.е. значение
21         //a*a и b*b.
22         int max = a*a + b*b - my_min( a*a, b*b );
23         printf("max (%d^2 , %d^2) ~ %d\n", a, b, max);
24         return 0;
25 }
```

Локальность аргументов Имя аргумента в функции никакого отношения не имеет к точке вызова функции. При изменении значения аргумента, значению переменной передаваемой в качестве аргумента не меняется. Совпадение или не совпадение имен аргумента и передаваемой переменной ни на что не влияет.

Локальные переменные Из функции не видны переменные из точке вызова функции. Последние означает невозможность их изменения. Даже при совпадении имен! Напомню, что функции вызываются по значению.

1.5 Рекурсия

Простая рекурсия Когда функция вызывает сама себя, это называется *рекурсией*. Например, факториал можно вычислить так:

```
1  #include <math.h>
2
3  int my_fact(int a)
4  {
```

```
5           if( a < 0 )//Факториал от отрицательных числе
6                   return -1;//не существует.
7           if( a == 0 || a == 1)
8                   return 1;//!0 ~ 1 и !1 ~ 1.
9           //Имеем, что a>1.
10          //Вызываем сами себя с меньшим значением.
11          //Учитывая, что a-1>0, гарантировано сойдется.
12          return a * my_fact(a-1);
13  }
14
15  int main()
16  {
17          int a;
18          printf("a?:");
19          scanf( "%d", &a);
20          printf("!%d ~ %d\n", a, my_fact(a));
21          return 0;
22  }
```

Упражнение. Напишите соответствующий цикл.

Контрольные упражнения Далее следуют контрольные упражнения.

Упражнение. Напиши программу подчета наибольшего общего делителя как итеративно, так и рекурсивно.

Упражнение. Напиши функция быстрого возведения в степень.

Упражнение. Фибоначи.