Курс «Основы программирования»

Федорук Елена Владимировна ст. преподаватель каф РК-6 МГТУ им.Н.Э.Баумана

Лекция №7

Функции

Функция — это именованный фрагмент программы. Данные могут передаваться в функцию и функция может возвращать значение.

Программа на языке Си состоит из функций, произвольным образом упорядоченных в файле. Фактически функции могут быть размещены и в разных файлах. Концепция функции в языке Си покрывает все типы подпрограмм в других языках программирования: подпрограммы, функции и процедуры.

Обычно программы на языке Си состоят из большого числа небольших функций, а не из немногих больших.

Функции позволяют писать модульные программы, в которых каждая функция выполняет собственную задачу. Функции обычно невелики (страница или меньше, могут быть и однострочные).

Функции позволяют избежать дублирования кода в одной программе. Кроме того, несколько программ могут совместно использовать код функции.

Программы легче читаются, так как детали “скрыты” внутри функций.

Программы легче разрабатываются, поскольку функции позволяют разбить большие задачи на задачи меньшего размера, более простые для решения.

Определение функции

Определение функции может иметь одну из двух форм:

```
возвращаемый_тип имя_функции (список объявлений формальных параметров)
{
  объявления   /* тело функции */
  операторы
}

возвращаемый_тип имя_функции (список формальных параметорв)
объявление формальных параметров
{
  объявления     /* тело функции */
  операторы
```

}

Рекомендуется первая форма, соответствующая стандарту ANSI языка Си. Вторая форма использует стандарт K&R языка Си и обеспечивает совместимость с существующими программами на языке Си.

Тело функции – это составной оператор.

Именем функции является идентификатор. Подобно другим идентификаторам имя может содержать буквы, цифры, символ подчеркивания, но не может начинаться с цифры. Имя функции считается внешним идентификатором. Оно может использоваться внутри любой другой функции (или внутри определяемой функции). Количество значащих символов в имени зависит от системы. Рекомендуется использовать в именах строчные буквы.

В функцию может передаваться информация извне через аргументы, которые называются *формальными параметрами*. Кроме того, функция может возвращать значение в вызывающую ее функцию, которое называется *возвращаемым значением*. Тип возвращаемого значения определяет ключевое слово, стоящее перед именем функции в ее определении.

Важным различием между определением функции в стандарте ANSI (называемым также определением прототипа функции) и определением функции в стандарте K&R является то, что использование формы ANSI требует, чтобы при вызове функции ее аргументы были преобразованы к типу, ожидаемому функцией. Для того, чтобы эти соглашения выполнялись, вызову функции должно предшествовать определение прототипа функции или объявление прототипа функции. Если тип фактического параметра не может быть преобразован к требуемому типу аргумента, компилятор выдает сообщение об ошибке.

При использовании определения или объявления функции в форме K&R Си выполняются предполагаемые по умолчанию преобразования типа.

Пример 1

```
/*возвращает площадь круга, заданного радиусом*/
double area_circle(double radius)
 {
 return 3.14159*radius*radius;
 }
```

Объявление функций

Оператор *объявления функции* (объявление прототипа функции) имеет следующий вид:

`возвращаемый_тип имя_функции (список объявлений типов параметров);`

В стандарте ANSI языка Си типы возвращаемого значения функции и типы каждого из аргументов указываются в определении функции.

Пример 1

```
double area_circle(double rad)   /* это определение функции*/
{
...
}
```

Любая вызывающая функция должна быть проинформирована о типе возвращаемого значения и о типах каждого из аргументов путем использования оператора объявления функции (объявление прототипа функции). Если объявление прототипа функции задано, то компилятор проверяет правильность использования функции. В противном случае подобные ошибки дадут о себе знать только при выполнении программы. Если определение функции предшествует вызову, то объявление прототипа функции не требуется.

Объявление может быть сделано перед вызывающей функцией или внутри ее.

Пример 2

```
int main()
{
double area_circle(double); /* это объявление функции*/
...
area=area_circle(rad);
...
}
```

Объявление функции **`area_circle()`** известно только той функции, в которой оно сделано, то есть функции main().

Пример 3

```
double area_circle(double); /* и это объявление функции*/
int main ()
{
...
area=area_circle(rad);
...
}
```

Объявление известно всем функциям, расположенным в исходном файле после него. Если при определении функции она будет объявлена с другим типом возвращаемого значения (например, **`float`**), будет выдано сообщение об ошибке, состоящей в повторном объявлении функции.

В объявлении прототипа функции могут указываться необязательные (фиктивные) имена.

Для функций, не возвращающих никакого значения, используется тип данных (ключевое слово) **`void`**. Тогда при неумышленном использовании оператора:

```
X = f();
```

во время компиляции будет выдано сообщение, указывающее на ошибку в операции присваивания (правая часть операции присваивания имеет тип **void**).

Пример 4

```
#include <stdio.h>
int main( )
{
 double area_circle(double);   /* это объявление функции*/
 double rad = 50.5, area;

 area=area_circle(rad);
 printf("Area is %f\n",area);
}
/*возвращает площадь круга, заданного радиусом*/
double area_circle(double radius)   /* это определение функции*/
 {
 return 3.14159*radius*radius;
 }
```

Вызов функции

Вызов функции влечет выполнение содержащихся в теле функции операторов.

Функции должны быть объявлены перед их вызовом. В Стандарт ANSI языка Си объявления функции (называемом также объявлением прототипа функции) используется для контроля типов. Компилятор выполнит проверки того, что количество и типы параметров, переданные функции, правильны.

Если объявление типа функции опущено, компилятор будет использовать правило умолчания K&R и установит для нее тип **int**. Для функций стандарта K&R компилятор не выполняет контроля типов.

Пример 1

```
#include <stdio.h>
int main()
{
 void intro (void);     /* объявление прототипа функции*/

 intro();            /*вызов функции*/
...
}
```

```
void intro (void)       /*определение прототипа функции*/
{
  printf("Работает вызываемая фукция intro()");
}
```

В данном примере при вызове функции **intro()** функция **main()** является *вызывающей функцией*. При вызове функции **printf()** вызывающей функцией является **intro()**. Ключевое слово **void** в объявлении прототипа функции **intro()** показывает, что функция **intro()** не имеет параметров и не возвращает значение.

Аргументы функции

Фактический параметр — выражение, которое указывается в круглых скобках в вызове функции. Часто фактические параметры функции называют аргументами.

В вызове функции должно быть указано правильное количество фактических параметров. Желательно явно привести фактические параметры к требуемому типу. Для этого используется операция приведения **(type)**.

Компилятор стандарта ANSI языка Си проверяет согласованность фактических параметров вызова функции ее формальным параметрам в следующих случаях:

- если функция объявлена в программе перед ее вызовом;
- если функция определена в исходном файле перед вызовом.

Компилятор ANSI Си проверяет правильность вызова функции: правильно ли указано количество фактических параметров и их типы. Если необходимо, компилятор выполнит автоматическое преобразование фактических параметров вызова функции к соответствующему типу. Если преобразование не имеет смысла, во время компиляции выдается сообщение об ошибке.

В функцию передается значение аргумента или фактического параметра, которое присваивается соответствующему формальному параметру.

Пример 1

```
#include <stdio.h>
int main()
{
void print_growth (float, float);       /*объявление прототипа функции */
float  amount = 250.,                   /*начальный вклад */
```

```
    interest = .075;                    /* процент 7.5% */

print_growth (amount, interest);        /*вызов функции */
                                        /* amount, interest - фактические параметры*/
printf ("main: amount is %.2f \n", amount);
}

void print_growth (float val, float rate)  /*определение прототипа функции */
                                        /*val, rate -формальные параметры, при вызове принимают значения фактических параметров:*/
                                        /* val = 250., rate=.075 */
{
  val = (1+rate)* val;
  printf ("Value after 1 year: %.2f\n", val);
 }
```

Результат выполнения программы:

```
Value after 1 year: 268.75
main: amount is 250.00
```

При вызове в функцию **print_growth()** передаются значения фактических параметров: в данном случае значения переменных **amount** и **interest**. Эти значения присваиваются новым переменным (формальным параметрам) **val** и **rate**, имеющим свои собственные адреса в памяти. Значение переменных **amount** и **interest** не изменяются вызовом функции **print_growth()**.

Назначение стека при вызове функции

Стек — это область памяти, используемая выполняемой программой для временного запоминания значений. При вызове функции все значения фактических параметров функции помещаются на стек. Затем запоминаются значения существенных регистров, таких как, например, указатель текущей команды. Значения регистров восстанавливаются при завершении управления вызванной функцией. При передаче управления вызванной функции значения фактических параметров связываются с соответствующими областями памяти на стеке, выделенными под формальные параметры. Затем определяется расположение на стеке для локальных (автоматических) переменных. Этот набор параметров, регистров и автоматических переменных для вызова функции называется *кадром* (frame). Если вызванная функция выполняет вызов другой функции, на стек помещается еще один кадр. Только один кадр является активным в данный момент времени. Активная часть стека "растет" при вызове функции и "сокращается" при выходе из функции. Стековое пространство все время повторно используется, и это является причиной того, почему автоматические переменные имеют до инициализации

неопределенные значения. Они зависят от того, какие значения запоминались в соответствующем месте памяти до данного вызова функции.

Пример 1

```
int main()
{
  int x = 10, y = 100;
  f(5, 10);
. . .
}

double f (int x, int sum)
{
  double total;
  g(20.5);
. . .
}

float g(double val)
{
int count;
. . .
}
```

Показанная на рис. 1 диаграмма показывает гипотетическую систему построения стека. Компиляторы Си управляют стеком различными способами. Например, стек может расти от больших адресов к меньшим или наоборот; количество запоминаемых регистров зависит от системы; стековое пространство в некоторых системах может выделяться динамически по мере требований и т.д.

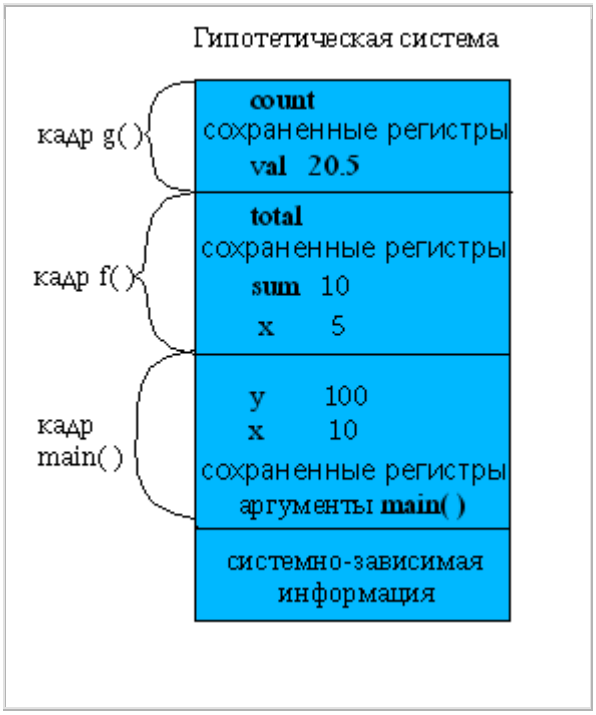

Рис. 1. Диаграмма построения стека

Оператор return

Функция может передавать одно значение обратно в вызывающую функцию.

Тип возвращаемого значения определяется в определении прототипа функции и объявляется в объявлении прототипа функции.

Оператора возврата return — оператор, с помощью которого функция возвращает значение в вызывающую функцию.

Синтаксис оператора возврата **return**:

return [выражение];

При выходе на оператор **return** выполняется возврат из функции так, что управление вновь передается в вызывающую функцию на оператор, следующий за вызовом. В случае если оператор **return** имеет выражение, значением вызова функции является значение этого выражения. Если оператор **return** не имеет выражения, значение вызова функции неопределено.

Функция может иметь более одного оператора **return**.

Пример 1

```
int func1(void)
{
...
if (n==0 || n==1)
  return 1;
else
  return 0;
}
```

Пример 2

```
void func2(void)
{
...
return ;
}
```

В прим. 1 функция **func1()** возвращает значение целого типа. В прим. 2 функция **func2()** не возвращает никакого значения.

Функция main() может использовать оператор **return** для возврата некоторого целого значения в родительский процесс.

Пример 3

```
#include <stdio.h>
int main ()
{
  int area_rect (int, int);
  int len = 50, width = 4, area;

  area = area_rect (len, width);
  printf ("Area is %d \n", area);
  return 0;
}

int area_rect (int l, int w)
{
/*площадь прямоуголника*/
  return l*w;
}
```

Рекурсивные функции

Функция в языке программирования Си может быть *рекурсивной*, т.е. может вызывать сама себя. Каждый раз, когда функция вызывает себя, на стек помещается новый кадр. Функция должна вызывать себя условно, например, внутри оператора if, иначе бесконечная рекурсия переполнит стек.

Пример 1

```
void printd (int n)
{
 int i;

 if (n < 0)
   {
    putchar ('-');
    n = -n;
   }
 if ( (i = n/10) != 0)
    printd(i);
 putchar(n % 10 + '0');
}
```