Тема : знакомство с C (Си)
ФУНКЦИИ, упражнения

Содержание: функции языка С упражнение 1 упражнение 2

- ## Функции

Мощность языка **C** во многом определяется легкостью и гибкостью в определении и использовании функций в **C**-программах. В отличие от других языков программирования высокого уровня в языке **C** нет деления на процедуры, подпрограммы и функции, здесь вся программа строится только из функций.

Определение

Функция - это совокупность объявлений и операторов, обычно предназначенная для решения определенной задачи. Каждая функция должна иметь имя, которое используется для ее объявления, определения и вызова. В любой программе на **C** должна быть функция с именем **main** (главная функция), именно с этой функции, в каком бы месте программы она не находилась, начинается выполнение программы.

При вызове функции ей при помощи аргументов (формальных параметров) могут быть переданы некоторые значения (фактические аргументы), используемые во время выполнения функции. Функция может возвращать некоторое (одно !) значение. Это возвращаемое значение и есть результат выполнения функции, который при выполнении программы подставляется в точку вызова функции, где бы этот вызов ни встретился. Допускается также использовать функции, не имеющие аргументов, и функции, не возвращающие никаких значений. Действие таких функций может состоять, например, в изменении значений некоторых переменных, выводе на печать некоторых текстов и т.п.

С использованием функций в языке **C** связаны три понятия –

- **определение функции** (описание действий, выполняемых функцией),
- **объявление функции** (задание формы обращения к функции) и
- **вызов функции**.

Определение функции задает тип возвращаемого значения, имя функции, типы и число формальных параметров, а также объявления переменных и операторы, называемые телом функции, и определяющие действие функции. В определении функции также может быть задан класс памяти.

Пример:

```
int eng (char r)
      {
         if (r>='A' && r<='R')
            return 1;
         else
            return 0;
      }
```

В данном примере определена функция с именем **eng**, имеющая один параметр с именем **r** и типом **char**. Функция возвращает целое значение, равное 1, если параметр функции является прописной буквой английского алфавита от A до R, или 0 в противном случае.

В языке **C** нет требования, чтобы определение функции обязательно предшествовало ее вызову. Определения используемых функций могут следовать за определением функции **main**, перед ним, или находится в другом файле.

Однако для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических параметров типам формальных параметров до вызова функции нужно поместить объявление (*прототип*) функции.

Объявление функции имеет такой же вид, что и определение функции, с той лишь разницей, что тело функции отсутствует, и имена формальных параметров тоже могут быть опущены. Прототип может иметь вид

```
int func1 (unsigned char r);
int func1 (unsigned char);
int func2 (char);
```

В программах на языке **C** широко используются, так называемые, библиотечные функции, т.е. функции предварительно разработанные и записанные в библиотеки. Прототипы библиотечных функций находятся в специальных заголовочных файлах, поставляемых вместе с библиотеками в составе систем программирования, и включаются в программу с помощью директивы **#include**.

Объявление параметров функции при ее определении может быть выполнено в так называемом "*старом стиле*" (конечно, если этот стиль поддерживается), при котором в скобках после имени функции следуют только имена параметров, а после скобок объявления типов параметров. Например, функция **eng** из предыдущего примера может быть определена следующим образом:

```
int eng (r)
 char r;
 { ...     /*  тело функции   */       ...  }
```

В соответствии с синтаксисом языка **C** определение функции имеет следующую форму:

[спецификатор-класса-памяти] [спецификатор-типа] имя-функции
([список-формальных-параметров])
{ тело-функции }

То есть

```
спецификатор-типа имя-функции (параметр1, параметр2, ...)
{
декларирование локальных переменных;
тело функции;
return значение функции;
}
```

Необязательный **спецификатор-класса-памяти** задает класс памяти функции, который может быть, например, **static**. (За информацией по классам памяти стоит обратиться к источникам, приведенным, например, в списке литературы.)

Спецификатор-типа функции задает тип возвращаемого значения и может задавать любой тип. Если **спецификатор-типа** не задан, то предполагается, что функция возвращает значение типа **int**.

Функция не может возвращать массив или функцию, но может возвращать указатель на любой тип, в том числе и на массив и на функцию. Тип возвращаемого значения, задаваемый в определении функции, должен соответствовать типу в объявлении этой функции.

Функция возвращает значение, если ее выполнение заканчивается оператором **return**, содержащим некоторое выражение. Указанное выражение вычисляется, преобразуется, если необходимо, к типу возвращаемого значения и возвращается в точку вызова функции в качестве результата. Если оператор **return** не содержит выражения или выполнение функции завершается после выполнения последнего ее оператора (без выполнения оператора **return**), то возвращаемое значение не определено. Для функций, не использующих возвращаемое значение, должен быть использован тип **void**, указывающий на отсутствие возвращаемого значения. Если функция определена как функция, возвращающая некоторое значение, а в операторе **return** при выходе из нее отсутствует выражение, то поведение вызывающей функции после передачи ей управления может быть непредсказуемым.

Список-формальных-параметров – это последовательность объявлений формальных параметров, разделенная запятыми. Формальные параметры – это переменные, используемые внутри тела функции и получающие значение при вызове функции путем копирования в них значений соответствующих фактических параметров. Если **список-формальных-параметров** заканчивается запятой с многоточием (,...), это означает, что число аргументов функции переменно. Однако предполагается, что функция имеет, по крайней мере, столько обязательных аргументов, сколько формальных параметров задано перед последней запятой в списке параметров. Такой функции может быть передано большее число аргументов, но над дополнительными аргументами не проводится контроль типов. В программах могут быть определены также макросы с переменным числом параметров, использующие многоточие

Если функция не использует параметров, то наличие круглых скобок обязательно, а вместо списка параметров рекомендуется указать слово **void**.

Порядок и типы формальных параметров должны быть одинаковыми в определении функции и во всех ее объявлениях. Типы фактических параметров при вызове функции должны быть совместимы с типами соответствующих формальных параметров. Тип формального параметра может быть любым основным типом, структурой, объединением, перечислением, указателем или массивом. Несоответствие типов фактических аргументов и формальных параметров может быть причиной неверной интерпретации.

Тело функции – это составной оператор, содержащий операторы, определяющие действие функции.

Все переменные, объявленные в теле функции, являются локальными. При вызове функции локальным переменным отводится память в стеке и производится их инициализация, если есть соответствующие операторы. Управление передается первому оператору тела функции и начинается выполнение функции, которое продолжается до тех пор, пока не встретится оператор **return** или последний оператор тела функции. Управление при этом возвращается в

точку, следующую за точкой вызова, а локальные переменные становятся недоступными. При новом вызове функции для локальных переменных память распределяется вновь, и поэтому старые значения локальных переменных теряются.

Параметры функции передаются **по значению (by value)** и могут рассматриваться как локальные переменные, для которых выделяется память при вызове функции и производится инициализация значениями фактических параметров. При выходе из функции значения этих переменных теряются. Поскольку передача параметров происходит по значению, в теле функции нельзя изменить значения переменных в вызывающей функции, являющихся фактическими параметрами. Однако, если в качестве параметра передать указатель на некоторую переменную, то используя операцию разадресации можно изменить значение этой переменной **(по ссылке – by reference)**.

Пример:

```
#include <stdio.h>

//функция main
int main (void)
{
int a,b;
int summa;

a=5; b=3;
//вычисление суммы

summa = a+b;

printf("\nKui liita 5 ja 3, on tulemuseks %d",summa);

return 0;
}
```

Предположим, что необходимо преобразовать программу, чтобы в дальнейшем можно было вы складывать различные числа. Перепишем программу, написав функцию, которая суммирует значения переданных параметров:

```
#include <stdio.h>

//описание функции
int liidaArvud(int a, int b)
{
int s;
s=a+b;
return (s);
}


//функция main
int main (void)
{
```

```
int summa;

//вызов функции  liidaArvud

summa = liidaArvud (5,3);

printf("\nKui liita 5 ja 3, on tulemuseks %d",summa);

return 0;
}
```

В ходе работы программы выводится на экран информация:

```
Kui liita 5 ja 3 on tulemuseks 8
```

В теле функции **main** декларируется одна целая переменная (int) с именем **summa**. Затем происходит вызов функции **liidaArvud** (**main** приостанавливает свою работу и передает управление функции **liidaArvud**) с заданными значениями 5 и 3, которые соответствуют параметрам функции int **a** и int **b**:

int liidaArvud (**int** a, **int** b);

summa= liidaArvud (**5**, **3**);

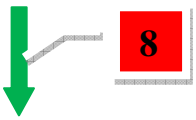
Функции **liidaArvud** передаются значения **5** и **3,** эти значения присваиваются формальным параметрам **int a** и **int b** соответственно. В функции задекларирована еще одна переменная **int s**, и присутствует операция **s = a + b;** переменной **s** присваивается значение суммы **a** и **b**, то есть **s** принимает значение **8**.

Предложение

```
return (s);
```

заканчивает выполнение функции **liidaArvud** и возвращает вычисленное значение вызывающей функции, в данном случае функции **main,** после этого функция **main** продолжает свою работу: выполняется операция, следующая за вызовом функции **liidaArvud**.

int liidaArvud (**int** a, **int** b);



summa = liidaArvud (**5**, **3**);

Вызывать теперь функцию можно многократно, задавая разные значения параметров.

```
…
{
int summa, summa_1;

//вызовы функции  liidaArvud
summa = liidaArvud (5,3);
…
summa_1 = liidaArvud (6,8);

…
}
```

Пример:

```
/*    Неправильное использование параметров    */
void change (int x, int y)
{    int k=x;
         x=y;
         y=k;
}
```

В данной функции значения переменных x и y, являющихся формальными параметрами, меняются местами, но поскольку эти переменные существуют только внутри функции change, значения фактических параметров, используемых при вызове функции, останутся неизменными. Для того чтобы менялись местами значения фактических аргументов можно использовать функцию, приведенную в следующем примере.

Пример: (использование указателей)

```
/*   Правильное использование параметров   */
     void change (int *x, int *y)
     {
       int k=*x;
           *x=*y;
           *y=k;
     }
```

При вызове такой функции в качестве фактических параметров должны быть использованы не значения переменных, а их адреса

```
change (&a,&b);
```

Необходимо еще раз обратить внимание на то, что, если вызов функции стоит перед определением самой функции, то надо задать прототип данной функции.

Объявление (прототип) функции имеет вид:

```
тип <имя функции>(список параметров);
```

Обратите внимание на то, что при описании функции после заголовка функции
тип <имя функции>(список параметров)
точку с запятой не ставят, а при объявлении функции точку с запятой ставят. Если прототип функции поставить перед **main**, то данная функция будет доступна всем функциям в файле.

Пример:

```
#include <stdio.h>

int main(void)
{
      void function(void);          // Объявление функции
      function();                   //  Вызов функции
return 0;
}


/* Описание функции  */
void function(void)                  // Заголовок функции
{                                    // Начало тела функции
      printf("Hello, World!!!\n");
}                                    //  Конец тела функции
```

- ## Упражнение 1

Разобрать, что делает следующая программа, добавить комментарии и попробовать ее запустить.

```
#include <stdio.h>

int liidaArvud(int a, int b)
{
  int s;
  s=a+b;
  return (s);
}

int main (void)
{
  int esimeneSumma, teineSumma, kolmasSumma;
  int esimeneLiidetav,teineLiidetav,kolmasLiidetav,neljasLiidetav;


  esimeneSumma = liidaArvud (5,3);
  printf("\nKui liita 5 ja 3, on tulemuseks %d",esimeneSumma);

  esimeneLiidetav=7;
  teineLiidetav=6;
```

```
  teineSumma=liidaArvud(esimeneLiidetav,teineLiidetav);
  printf("\nKui liita %d ja %d, on tulemuseks %d",esimeneLiidetav,
teineLiidetav, teineSumma);

  printf("\nSisesta yks arv: ");
  scanf("%d",&kolmasLiidetav);
  printf("\nSisesta teine arv: ");
  scanf("%d", &neljasLiidetav);

  kolmasSumma=liidaArvud(kolmasLiidetav,neljasLiidetav);
  printf("\nKui liita %d ja %d, on tulemuseks %d\n",kolmasLiidetav,
neljasLiidetav, kolmasSumma);


  return 0;
}
```

Пример:

Рассмотрим функцию, которая ничего не возвращает, но принимает параметры: строковую переменную и целое число. В ходе работы функции на экран выводится некоторая информация.

```
void vanusEkraanile(char s[], int v)
{
    printf("\nTere %s, sa oled %d aastat vana\n", s, v);
}
```

Можно использовать и указатель `char *s`:

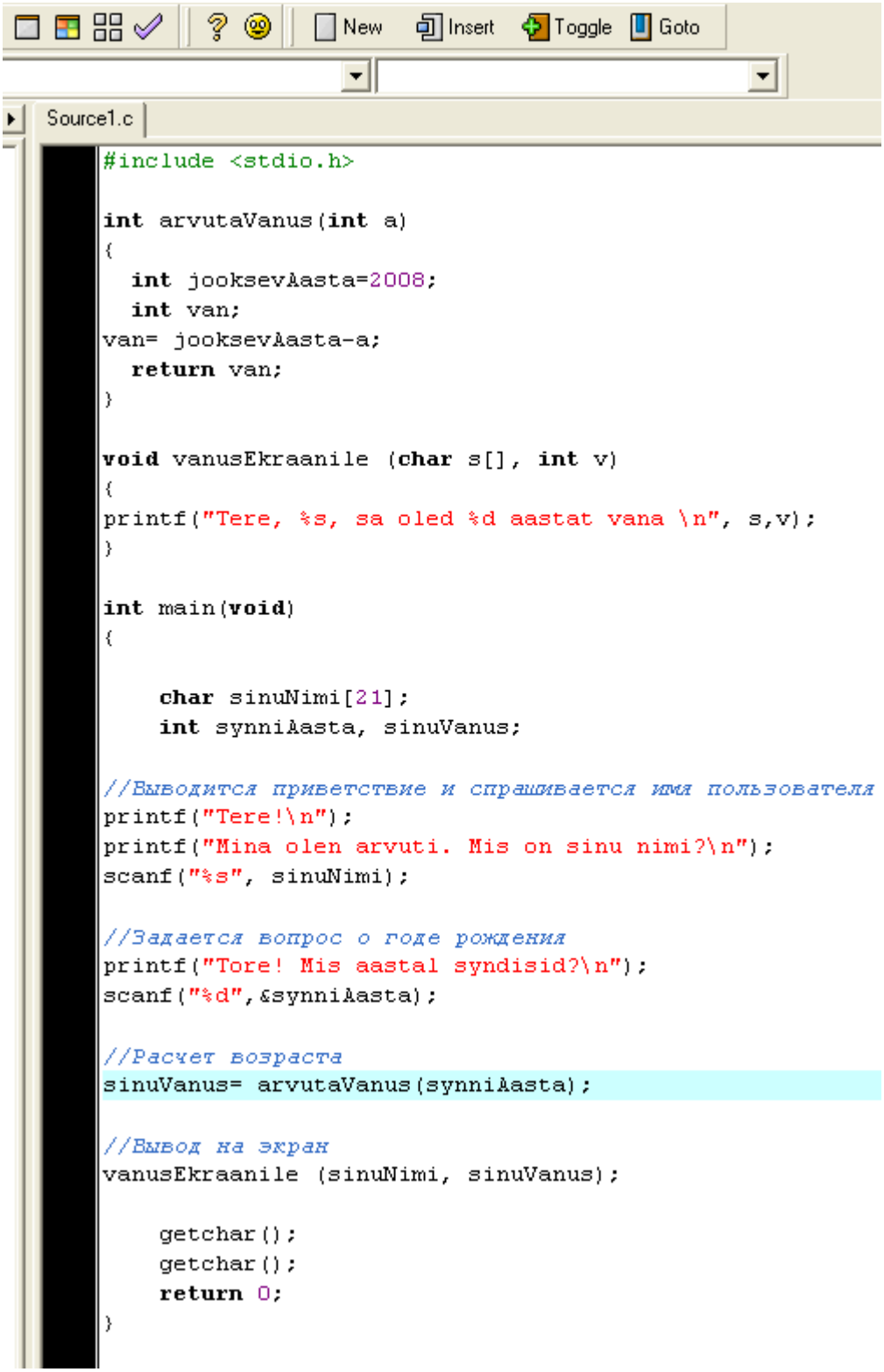
```
void vanusEkraanile(char *s, int v)
{
    printf("\nTere %s, sa oled %d aastat vana\n", s, v);
}
```

К этой функции обращаемся следующим образом:

```
vanusEkraanile(sinuNimi,sinuVanus);
```

- ## Упражнение 2

Рассмотрим опять пример, в котором программа просит пользователя ввести имя, год рождения и рассчитывает возраст пользователя. Выводятся на экран имя и вычисленный возраст.



Скомпилируем программу в **Terminal-е**

```
gcc -o Source1 Source1.c
```

Запуск программы:

```
./Source1
```

Используя программу из занятия (**часть 1**) и пример, приведенный выше, создать следующую программу.

- В ходе работы программы происходит следующее: (простой текст – это то, что выдает компьютер, **жирным текстом** выделено, то, что вводит пользователь):

```
Tere, mis on sinu nimi?
Juri
Juri! Liidame kaks arvu
Sisesta esimene arv: 15
Sisesta teine arv: 7
Juri, arvude 15 ja 7 summa on 22

Proovime jagada
Sisesta jagatav: 15
Sisesta jagaja: 7
Kui jagame 15.00 arvuga 7.00, on tulemuseks 2.143
```

- **NB!** Создать функции, которые выполняют следующие действия:
  a. Сложение двух целых чисел (int) , и возврат результата (return)
  b. Вывод на экран строки символов (char[]), содержащей три целых числа (int)
  c. Деление одного числа плавающего типа на другое (double), и возврат результата (return)
  d. Вывод на экран строки символов (char[]), содержащей три числа плавающего типа (double)
- Затем создать функцию (например, с именем **arvutaJaVäljasta**), которая будет использовать созданные ранее функции (пункты **a** и **b**). Программа должна работать следующим образом:
  - **main** вызывает **arvutaJaVäljasta**
  - **arvutaJaVäljasta** вызывает функции сложения и вывода
  - создать аналогичную функцию для пунктов **c** и **d**.

Отчет оформляется в MS Word (или в другом текстовом редакторе), представляется в распечатанном виде. Отчет должен содержать необходимое для понимания описание работы. Представляемые программы должны быть прокомментированы.

Использованы материалы:

- ***Громов, Титаренко***. Программирование на языке С.
- ***Керниган Б., Ритчи Д. Язык С.***
- ***Материалы*** *Helena Kruus (MSc)*
- ***The cplusplus.com tutorial.*** *Complete C++ language tutorial* http://www.cplusplus.com/doc/tutorial/index.html *[5.11.2010]*
- ***The GNU C library*** http://www.gnu.org/software/libc/manual/html_node/ *[5.11.2010]*

Марина Брик
Составлено: 15.11.2007
Обновлено: 28.10.2008
Обновлено: 7.11.2010