Создание и использование функций в языке Си

Функция

Функция – это именованный логически целостный фрагмент программы. Данные могут передаваться в функцию и функция может возвращать значение.
Программа на языке Си состоит из функций - базовых блоков в которых выполняются все операции. Функции могут быть размещены как в одном файле программы, так и в разных файлах. Концепция функции в языке Си покрывает все типы подпрограмм в других языках программирования: функции и процедуры.

Функция

Можно перечислить основные удобства использования функций:
функции позволяют писать модульные программы;
функции позволяют избежать дублирования кода в одной программе;
несколько программ могут совместно использовать код одной функции;
программы легче читаются, так как детали “скрыты” внутри функций;
программы легче пишутся, поскольку функции позволяют разбить большие задачи на более простые задачи.

Функция

Стандартный вид объявления функций следующий:

спецификатор_типа имя_функции (список параметров)

```
{
    тело функции
}
```

Спецификатор_типа определяет тип возвращаемого функцией значения с помощью оператора return. Это может быть любой допустимый тип. Если тип не указан, предполагается, что функция возвращает

Функция

Список параметров - это разделенный запятыми список переменных, получающий значение аргументов при вызове функции. Функция может быть без параметров и в таком случае список параметров должен состоять из ключевого слова **void.**

Имя функции является **идентификатором**. Подобно другим идентификаторам имя может содержать буквы, цифры, символ подчеркивания, но не может начинаться с цифры.

Функция

Имя функции считается внешним идентификатором; оно может использоваться внутри любой другой функции (или внутри определяемой функции).

Пример:

```
double area_circle(double radius)
{
return 3.14159*radius*radius;
}
```

Функция

Оператор объявления прототипа функции имеет следующий вид:
возвращаемый_тип имя_функции (список объявлений типов параметров);
Пример: 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 прототипа функции.

Функция

Если объявление прототипа функции задано, то компилятор проверяет правильность использования функции. В противном случае подобные ошибка 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:

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

Объявление функции **area_circle**() известно только той функции, в которой оно сделано, то есть функции **main**(). Если расположить строку с объявлением функции перед функцией **main**(), то эта функция будет известна всем функциям, расположенным в исходном файле после него.

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

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 void. Для такой функции при неумышленном использовании оператора присваивания:
X = f();
во время компиляции будет выдано сообщение, указывающее на ошибку в операции присваивания.

Пример:

```
#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);
}
double area_circle(double radius)
/* это определение функции*/
{
    return 3.14159*radius*radius;
}
```

- Реализация программы с объявлением функции

```c
#include <stdio.h>
double area_circle(double radius);
int main( )
{
    double area_circle(double);    /* это объявление функции*/
    double rad = 50.5, area;
    area=area_circle(rad);
    printf("Area is %f\n",area);
    return 0;
}
double area_circle(double radius)    /* это определение функции*/
{
    return 3.14159*radius*radius;
}
```

```
"H:\HM-MAIN\Kafedra\
Area is 8011.839898

Process returned 0 (0x0)   execution time : 0.219 s
Press any key to continue.
```

- Реализация программы без предварительного объявления функции

Start here | Exs_3_1_функция площадь круга.c

```
1   #include <stdio.h>
2   int main( )
3   {
4       double area_circle(double);    /* это объявление функции*/
5       double rad = 50.5, area;
6       area=area_circle(rad);
7       printf("Area is %f\n",area);
8       return 0;
9   }
10  double area_circle(double radius)    /* это определение функции*/
11  {
12      return 3.14159*radius*radius;
13  }
14
```

"H:\HM-MAIN\Kafedra\

```
Area is 8011.839898

Process returned 0 (0x0)   execution time : 0.250 s
Press any key to continue.
```

- Пример функции перед функцией main()

```c
#include <stdio.h>
#include <math.h>
float fun(float a, float b)
{ return sin(a)*b+2*b*a-8 ; }
int main()
{
    printf("fun=%f\n",fun (5,8));
        return 0;
}
```

```
fun=64.328606

Process returned 0 (0x0)   execution time
Press any key to continue.
```

- Пример функции внутри функции main()

```c
#include <stdio.h>
#include <math.h>
int main()
{
float fun(float a, float b)
{ return sin(a)*b+2*b*a-8 ; }
    printf("fun=%f\n",fun (5,8));
        return 0;
}
```

```
fun=64.328606

Process returned 0 (0x0)   execution
Press any key to continue.
```

- Вообще оператор return имеет два назначения. Во-первых, немедленный выход из функции. То есть он осуществляет выход в вызывавший функцию код. Во-вторых, может использоваться для возврата значения. Когда используется оператор return в main(), программа возвращает код завершения вызывавшему процессу (операционной системе). Возвращаемое значение должно быть целого типа. Большинство операционных систем, трактуют 0 как нормальное завершение программы. Остальные значения воспринимаются как ошибки. Если не определено возвращаемое значение, то в операционную систему будет передано неизвестное значение. Поэтому гораздо полезнее использовать оператор return.

Пример. Табуляц
функции (из
практического
задания) без
использования
функции СИ.

```c
#include <stdio.h>
int main()
{
float s,e,h;
printf("Enter a lower number s : ");scanf("%f",&s);
printf("Enter a larger number e : ");scanf("%f",&e);
printf("Enter step tab h : ");scanf("%f",&h);
printf("Table:\n");
float pi = 3.142;
float x = (s*pi)/180;
while(x <= (e*pi)/180)
{
float y = pow(sin(7*pi/8 - 2*x),2);
printf("x : %f ;\t y : %f\n",(x*180)/pi,y);
x = x + (h*pi)/180;
}
return 0;
}
```

```
Enter a lower number s : 5
Enter a larger number e : 70
Enter step tab h : 5
Table:
x : 5.000000 ;    y : 0.288388
x : 10.000000 ;   y : 0.456112
x : 15.000001 ;   y : 0.629131
x : 20.000000 ;   y : 0.786570
x : 25.000000 ;   y : 0.909436
x : 30.000001 ;   y : 0.982906
x : 35.000001 ;   y : 0.998115
x : 40.000000 ;   y : 0.953228
x : 45.000000 ;   y : 0.853661
x : 50.000000 ;   y : 0.711427
x : 54.999999 ;   y : 0.543685
x : 60.000002 ;   y : 0.370672
x : 65.000002 ;   y : 0.213263

Process returned 0 (0x0)   execution time : 12.469 s
Press any key to continue.
```

Пример. Табуляция функции (из практического задания) с использованием функции

```c
int main()
{
float s,e,h;
printf("Enter a lower number s : ");scanf("%f",&s);
printf("Enter a larger number e : ");scanf("%f",&e);
printf("Enter step tab h : ");scanf("%f",&h);
printf("Table:\n");
float pi = 3.142;
float x = (s*pi)/180;
float f(float x);
while(x <= (e*pi)/180)
{
float y = f(x);
printf("x : %f ;\t y : %f\n",(x*180)/pi,y);
x = x + (h*pi)/180;
}
return 0;
}

float f(float x)    /* это определение функции*/
{
    float pi = 3.142;
    return pow(sin(7*pi/8 - 2*x),2);
}
```

```
Enter a lower number s : 5
Enter a larger number e : 70
Enter step tab h : 5
Table:
x : 5.000000 ;   y : 0.288388
x : 10.000000 ;  y : 0.456112
x : 15.000001 ;  y : 0.629131
x : 20.000000 ;  y : 0.786570
x : 25.000000 ;  y : 0.909436
x : 30.000001 ;  y : 0.982906
x : 35.000001 ;  y : 0.998115
x : 40.000000 ;  y : 0.953228
x : 45.000000 ;  y : 0.853661
x : 50.000000 ;  y : 0.711427
x : 54.999999 ;  y : 0.543685
x : 60.000002 ;  y : 0.370672
x : 65.000002 ;  y : 0.213263

Process returned 0 (0x0)   execu
Press any key to continue.
```

- **Замечания к построению алгоритма нахождения корня**

2. МЕТОДЫ НАХОЖДЕНИЯ КОРНЕЙ УРАВНЕНИЙ

Очень часто на практике приходится решать уравнения вида $f(x)=0$, где функция $f(x)$ определена и непрерывна в интервале $a \le x \le b$. Под решением уравнения понимают нахождение некоторого значения x_0, которое обращает функцию $f(x)$ в ноль. Значение x_0 и называется корнем уравнения $f(x)=0$.

- **Замечания к построению алгоритма нахождения корня**

2.1. Метод половинного деления

Метод половинного деления, или метод деления отрезка пополам (дихотомии), чрезвычайно прост, и его алгоритм легко реализуется на ЭВМ.

Пусть необходимо решить уравнение $f(x) = 0$, где функция $f(x)$ непрерывна на отрезке $[a; b]$, и только один корень x_0 заключен в том же интервале. Таким образом, функция $f(x)$ будет знакопеременной на концах отрезка $[a; b]$ (рис. 2.1). Математически это можно записать как $f(a)\cdot f(b)<0$. Разделим отрезок $[a; b]$ пополам, т. е. найдем $x = \frac{a+b}{2}$ и вычислим значение функции $f(x)$ в этой точке.

Если окажется, что $f(x) = 0$, то x – корень уравнения. Если $f(x)\neq 0$, то выбираем ту половину отрезка $[a; x]$ или $[x; b]$, на концах которой функция $f(x)$ имеет противоположные знаки. На рис. 2.1 это отрезок $[x; b]$. Половина отрезка, не содержащая корня $[a; x]$, отбрасывается. Это означает, что левая граница интервала перемещается в точку деления пополам ($a=x$).

- **Замечания к построению алгоритма нахождения корня**

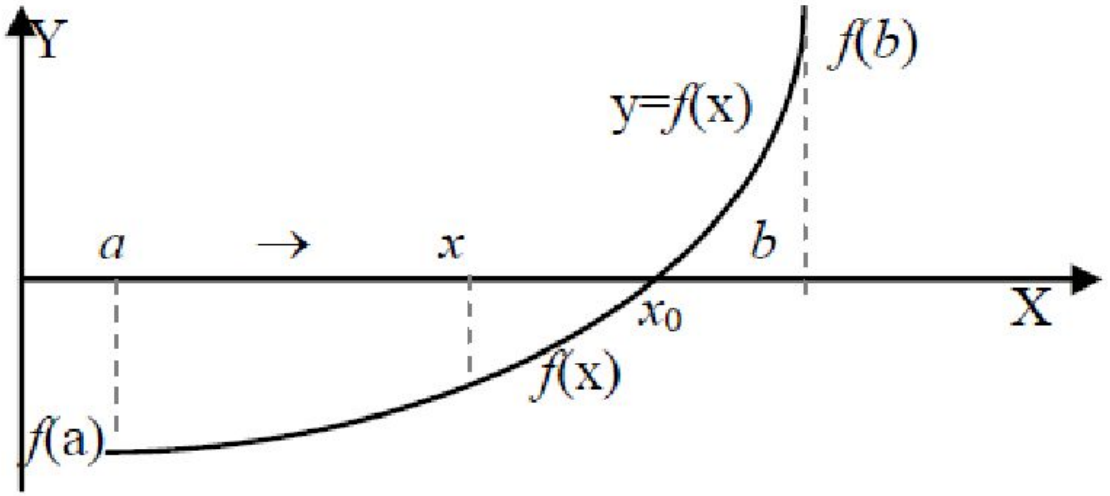

Рис. 2.1. *Геометрическое представление метода половинного деления*

При повторном делении производятся те же самые операции: новый отрезок $[a; b]$ делится пополам, вычисляется значение функции в точке деления $f(x)$ и определяется отрезок, содержащий истинный корень x_0. Процесс деления продолжают до тех пор, пока длина отрезка, содержащего корень, не станет меньше некоторого наперед заданного числа ε (точности) или пока значение функции в точке деления $y=f(x)$ превышает ε по абсолютной величине.

- **Замечания к построению алгоритма нахождения корня**

Метод Ньютона (метод касательных)

Пусть уравнение $f(x)=0$ имеет один корень на отрезке $[a; b]$, а его первая и вторая производные $f'(x)$ и $f''(x)$ определены, непрерывны и сохраняют постоянные знаки в этом интервале.

Геометрически метод Ньютона эквивалентен замене небольшой дуги кривой $y = f(x)$ касательной в некоторой точке кривой (рис. 2.3).

Выберем в качестве первого приближенного значения корня точку $x_1=b$, для которой выполняется условие $f(x_1)\cdot f''(x_1)>0$, т.е. значения функции и ее второй производной в точке x_1 имеют одинаковые знаки. Напомним, что знак второй производной от функции $f(x)$ определяет выпуклость кривой: если $f''(x)$ положительна, то график функции имеет выпуклость вниз, как это изображено на рис. 2.3.

Через точку R1 с координатами $[x_1, f(x_1)]$ проведем касательную к кривой $y = f(x)$. В качестве второго приближения x_2 корня возьмем абсциссу точки пересечения этой касательной с осью Ox. Через точку R_2 снова проведем касательную, абсцисса точки пересечения которой даст нам следующее приближение x_3 корня, и т. д.

- **Замечания к построению алгоритма нахождения корня**

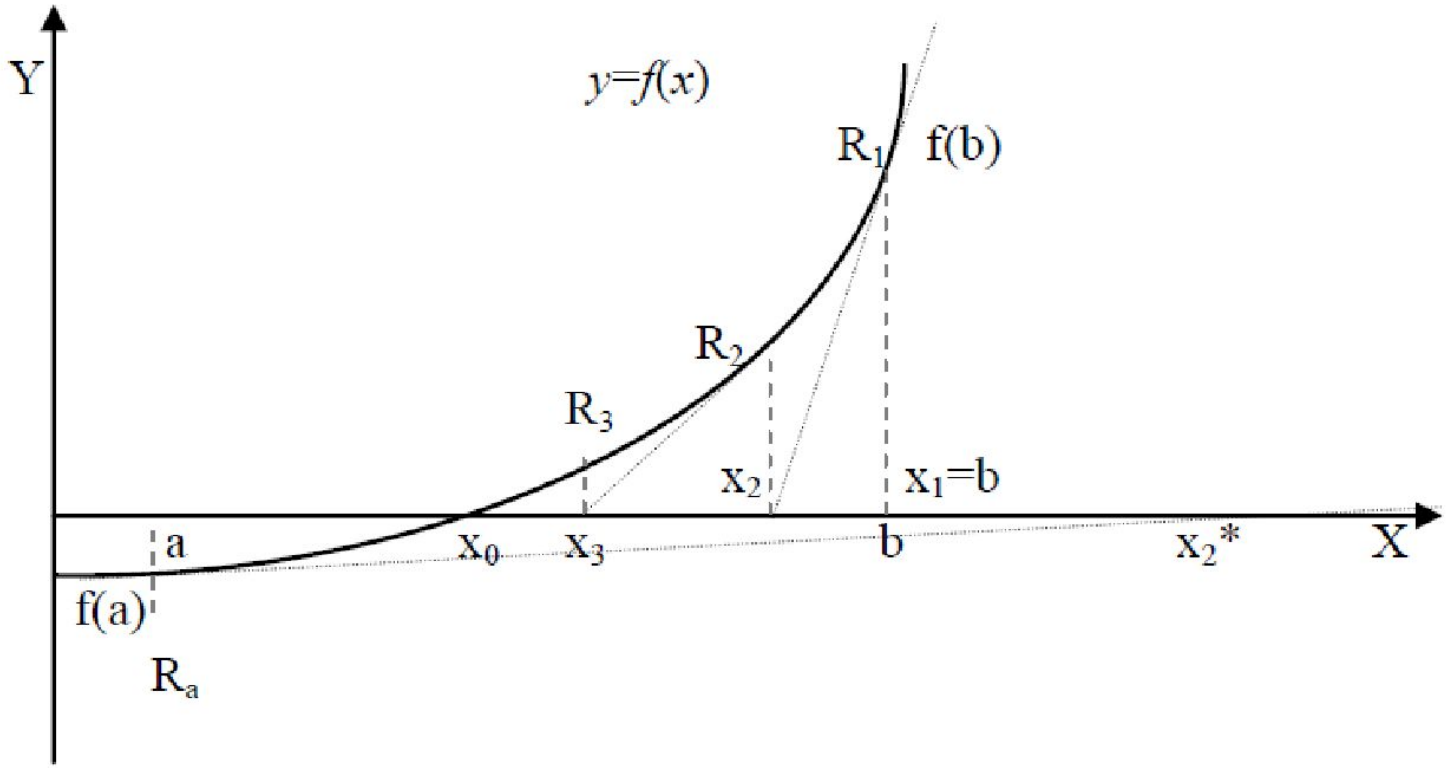

Рис. 2.3. *Геометрическое представление метода Ньютона*

- **Замечания к построению алгоритма нахождения корня**

Уравнение касательной, проходящей через точку x, имеет вид

$$y - f(x_1) = f'(x_1)\cdot(x - x_1). \qquad (2.2)$$

Полагая $y=0$, а $x=x_2$, найдем абсциссу x точки пересечения касательной с осью Ox:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}. \qquad (2.3)$$

Следующее приближение x_3 находим по формуле:

$$x_3 = x_2 - f(x_2) / f'(x_2). \qquad (2.4)$$

Применяя формулы уточнения значения корня многократно, получим общий вид рекуррентной формулы метода касательных:

$$x_{n+1} = x_n - f(x_n) / f'(x_n). \qquad (2.5)$$

Процесс вычислений заканчивают обычно по условию

$$|x_{n+1} - x_n| < \varepsilon. \qquad (2.6)$$

При значениях $\varepsilon < 10^{-4}$ погрешность вычисления корня приблизительно равна абсолютной погрешности ε.