Catalog

FUNCTIONS AND POINTERS

INTRODUCTION TO FUNCTION

Function is defined as the block of organized, reusable code that is used to perform the specific action. A function is a subprogram of one or more statements that performs a specific task when called.

Advantages of Functions:

1. Code reusability
2. Better readability
3. Reduction in code redundancy
4. Easy to debug & test.

Classification of functions:

a) Based on who develops the function
b) Based on the function prototype

Based on the function prototype

Function prototype is a declaration statement that identify function with function name, data type, a list of a arguments

Based on who develops the function

There are two types.

- Library functions
- User-defined functions

Library(Pre-defined) function	User defined function
Contains Pre-defined set of functions	The user defined the functions
User cannot understand the internal working	User can understand internal working
Source code is not visible	Source code is visible
User cannot modify the function	User can modify the function
Example: printf( ), scanf( ).	Example: sum() , square( )

Library Vs User-defined function

Elements of user defined function (or) Steps in writing a function in a program

1. Function Declaration (Prototype declaration)
2. Function Call
3. Function Definition

FUNCTION PROTOTYPE

Function prototype is a declaration statement that identifies function with function name, data type, a list of a arguments. All the function need to be declared before they are used. (i.e. called)

Syntax:

returntype functionname (parameter list);

- Return type – data type of return value. It can be int, float, double, char, void etc.
- Function name – name of the function
- Parameter type list –It is a comma separated list of parameter types.

Example:

int add(int a, int b);

Function declaration must be terminated with a semicolon(;).

Types of function prototypes:

1. Function with no arguments and no return values
2. Function with arguments and no return values
3. Function with arguments and one return values
4. Function with no arguments and with return values

Prototype 1: Function with no arguments and no return values

- This function doesn‟t accept any input and doesn‟t return any result.
- These are not flexible.

Program

```
#include<stdio.h>
#include<conio.h>
void  show(); //function prototype
void main()
{
        show( );            //function call
        getch();
}
```

```
void show( ) //function definition
{
        printf("Hai \n");
}
```

Output:

```
Hai
```

Prototype 2: Function with arguments and no return values

Arguments are passed through the calling function. The called function operates on the values but no result is sent back.

Program

```
#include<stdio.h>
#include<conio.h>
void show(int);
void main()
{
        int a;
        printf("Enter the value for a \n");
        scanf("%d", &a);
        show(a);
        getch();
}
void show(int x)
{
        printf("Value =%d", x);
}
```

Output:

```
Enter the value for a: 10
Value = 10
```

Prototype 3: Function with arguments and return values

- Arguments are passed through the calling function
- The called function operates on the values.

The result is returned back to the calling function.

Program

```
#include<stdio.h>
#include<conio.h>
float circlearea(int);
void main()
{
        int r;
        float area;
        printf("Enter the radius \n");
        scanf("%d",&r);
        area=circlearea(r);
        printf("Area of a circle =%d\n", area);
        getch();
}
int circlearea(int r1)
{
        return 3.14 * r1 * r1;
}
```

Output:

```
Enter the radius
2
Area of circle = 12.000
```

Prototype 4: Function with no arguments and with return values

- This function doesn‟t accept any input and doesn‟t return any result.
- The result is returned back to the calling function.

Program

```
#include<stdio.h>
#include<conio.h>
float circlearea();
void main()
```

```
        {
                float area;
                area=circlearea();
                printf("Area of a circle =%d\n", area);
                getch();
        }
        int circlearea()
        {
                int r=2;
                return 3.14 * r * r;
        }
```

Output:

```
        Enter the radius
        2
        Area of circle = 12.000
```

FUNCTION DEFINITION

It is also known as function implementation. When the function is defined, space is allocated for that function in memory.

Syntax

```
returntype functionname (parameter list)
{
statements;
return (value);
}
```

Example

```
int abc(int, int, int)  // Function declaration
void main()
{
        int x,y,z;
        abc(x,y,z) // Function Call
        …
        …
}
int abc(int i, int j, int k)  // Function definition
{
        …….
        ….
        return (value);
}
```

Every function definition consists of 2 parts:

a) Header of the function
b) Body of the function

a) Header of the function

The header of the function is not terminated with a semicolon. ***The return type and the number & types of parameters must be same*** in both function header & function declaration.

Syntax:

returntype functionname (parameter list)

Where,

- Return type – data type of return value. It can be int, float, double, char, void etc.
- Function name – name of the function
- Parameter type list –It is a comma separated list of parameter types.

b) Body of the function

- It consists of a set of statements enclosed within curly braces.
- The return statement is used to return the result of the called function to the calling function.

Program:

```
#include<stdio.h>
#include<conio.h>
float circlearea(int);                //function prototype
void main()
{
        int r;
        float area;
        printf("Enter the radius \n");
        scanf("%d",&r);
        area=circlearea(r); //function call
```

```
          printf("Area of circle =%f\n", area);
          getch();
     }
     float circlearea(int r1)
     {
          return 3.14 * r1 * r1;          //function definition
     }
```

Output:

```
     Enter the radius
     2
     Area of circle = 12.000
```

FUNCTION CALL

Function call is used to invoke the function. So the program control is transferred to that function. A function can be called by using its name & actual parameters.

Function call should be terminated by a semicolon (;).

Syntax:

```
Functionname(argumentlist);
```

Example

```
int abc(int, int, int)  // Function declaration
void main()
{
        int x,y,z;
        abc(x,y,z) // Function Call
        …
        …
}
int abc(int i, int j, int k)  // Function definition
{
        …….
        ….
        return (value);
}
```

Calling function – The function that calls a function is known as a calling function.

Called function – The function that has been called is known as a called function.

Actual arguments – The arguments of the calling function are called as actual arguments.

Formal arguments – The arguments of called function are called as formal arguments.

Steps for function Execution:

1. After the execution of the function call statement, the program control is transferred to the called function.
2. The execution of the calling function is suspended and the called function starts execution.
3. After the execution of the called function, the program control returns to the calling

function and the calling function resumes its execution.

Program:

```
#include<stdio.h>
#include<conio.h>
float circlearea(int);              //function prototype
void main()
{
        int r;
        float area;
        printf("Enter the radius \n");
        scanf("%d",&r);
        area=circlearea(r); //function call
        printf("Area of a circle =%f\n", area);
        getch();
}
 float circlearea(int r1)
{
        return 3.14 * r1 * r1;          //function definition
}
```

Output:

```
Enter the radius
2
Area of circle = 12.000
```

BUILT IN FUNCTIONS (STRING FUNCTIONS, MATH FUNCTIONS):

Library functions are predefined functions. These functions are already developed by someone and are available to the user for use.

MATHEMATICAL FUNCTIONS

These are defined in math.h header file. Math functions perform the mathematical calculations. Some of the built-in math functions are,

S.No.	Function & Description

1	**double acos(double x)** Returns the arc cosine of x in radians.
2	**double asin(double x)** Returns the arc sine of x in radians.
3	**double atan(double x)** Returns the arc tangent of x in radians.
4	**double atan2(double y, double x)** Returns the arc tangent in radians of y/x.
5	**double cos(double x)** Returns the cosine of a radian angle x.
6	**double cosh(double x)** Returns the hyperbolic cosine of x.
7	**double sin(double x)** Returns the sine of a radian angle x.
8	**double sinh(double x)** Returns the hyperbolic sine of x.

9	**double tanh(double x)** Returns the hyperbolic tangent of x.
10	**double exp(double x)** Returns the value of **e** raised to the xth power.
11	**double frexp(double x, int *exponent)** The returned value is the mantissa and the integer pointed to by exponent is the exponent. The resultant value is x = mantissa * 2 ^ exponent.
12	**double ldexp(double x, int exponent)** Returns **x** multiplied by 2 raised to the power of exponent.
13	**double log(double x)** Returns the natural logarithm (base-e logarithm) of **x**.
14	**double log10(double x)** Returns the common logarithm (base-10 logarithm) of **x**.
15	**double modf(double x, double *integer)** The returned value is the fraction component (part after the decimal), and sets integer to the integer component.
16	**double pow(double x, double y)** Returns x raised to the power of **y**.
17	**double sqrt(double x)** Returns the square root of **x**.
18	**double ceil(double x)** Returns the smallest integer value greater than or equal to **x**.
19	**double fabs(double x)** Returns the absolute value of **x**.
20	**double floor(double x)** Returns the largest integer value less than or equal to **x**.
21	**double fmod(double x, double y)** Returns the remainder of x divided by **y**.

Mathematical Functions

Program:

```
#include<stdio.h>
#include <math.h>
#include<conio.h>
void main()
{
    printf("\n%f",ceil(3.6));
    printf("\n%f",ceil(6.3));
    printf("\n%f",floor(3.6));
    printf("\n%f",floor(7.2));
    printf("\n%f",sqrt(16));
    printf("\n%f",sqrt(7));
    printf("\n%f",pow(2,4));
    printf("\n%f",pow(3,3));
    printf("\n%d",abs(-12));
    return 0;
}
```

Output:

```
4.000000
4.000000
3.000000
3.000000
4.000000
2.645751
16.000000
27.000000
12
```

STRING FUNCTIONS

String operations (length, compare, concatenate, copy) Refer Section 2.4 in pageno: 81.

strlwr() function

It converts all the uppercase characters in that string to lowercase characters.

Syntax

```
strlwr(string_name);
```

Program

```
str[10]= "HELLO";
strlwr(str);
puts(str);
```

Output:

```
hello
```

strupr() function

It converts all the lowercase characters in that string to uppercase characters.

Syntax

```
strupr(string_name);
```

Example

```
str[10]= "HEllo";
strupr(str);
puts(str);
```

Output:

```
HELLO
```

strrev() function

It is used to reverse the string.

Syntax

```
strrev(string_name);
```

Example

```
str[10]= "HELLO";
strrev(str);
```

```
puts(str);
```

Output:

```
OLLEH
```

Program for string reverse:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main ()
    {
      char str[20] = "Computer";
      printf("Given string = %s\n", str);
      printf ("The reverse string = %s", strrev(str));
    getch();
}
```

Output:

```
Given string = Computer
The reverse string = retupmoC
```

RECURSION

Recursion is defined as the function that calls itself repeatedly until condition is reached. But while using **recursion**, programmers need to be careful to define an exit condition from the function; otherwise it will go into an infinite loop.

Syntax:

```
Function1()
{
        Function1();
}
```

Example:

Calculating the factorial of a number

Fact (n)= n*fact(n-1);

6! = 6*fact(n);

6! = 6 *5*fact(4)

6! = 6 * 5 * 4 *fact(3)

6! = 6 * 5 * 4 * 3 *fact(2)

6! =6 *5 * 4 * 3 *2 * fact(1)

6! = 6 *5 * 4 * 3 *2 * 1

6!=120

Advantage of recursion

- Recursion makes program elegant and cleaner.
- All algorithms can be defined recursively which makes it easier to visualize and prove.
- Reduce unnecessary calling of function
- Easy to solve complex problems

Direct Recursion:

A function is directly recursive if it calls itself.

```
A( )
{
        ….
        A( ); // call to itself
        ….
}
```

Indirect Recursion:

Function calls another function, which in turn calls the original function.

```
A( )
{
        …
        B( );
        …
}
B( )
{
        …
        A( );// function B calls A
        …
}
```

Linear Recursion - It makes only one recursive call.

Binary Recursion - It calls itself twice.

N-ary recursion - It calls itself n times.

Program 1 : Find factorial using recursion

```
#include<stdio.h>
#include<conio.h>
int fact(int);
void main()
{
        int n, Result;
        printf("\n Enter any number:");
        scanf("%d", &n);
        Result = fact(n);
        printf ("Factorial value = %d", Result);
        getch();
}
int fact (int x)
{
        if (x == 0)
                return 1;
        else
                return x * fact(x – 1);
}
```

Output:

```
Enter any number: 4
Factorial value = 24
```

Program 2 : Find the GCD of Two Positive Integer Numbers

```
#include<stdio.h>
#include<conio.h>
int gcd (int a, int b)
void main ()
{
    int a, b;
    printf("\n Enter the two numbers:");
    scanf ("%d%d", &a, &b);
```

POINTERS

Pointer is a variable that stores the address of a variable or a function. Pointer is a derived data type. It is declared same as other variables but prior to variable name an asterisk „*‟operator is used.

Benefits of Pointers

- Reduce the length of the program.
- Support dynamic memory management.
- Pointers save memory space.
- More efficient in handling arrays.
- Execution speed increased.
- Used to return multiple values from a function.

Declaring a Pointer

The pointer variable is declared by using * operator.

syntax

```
datatype  * pointervariable;
```

Example:

```
char *ptr;
float *P1;
```

Pointer Initialization

The process of assigning the address of a variable to a pointer variable is known as initialization. This is done through (&) ampersand operator.

Syntax:

```
Pointer variable = & variablename;
```

Example:

```
int n, *p;
 p = &n;
```

POINTER OPERATORS

Pointer is a variable that stores the address of a variable or a function. There are 2 pointer operators in C. They are,

1. Referencing operator
2. Dereferencing operator

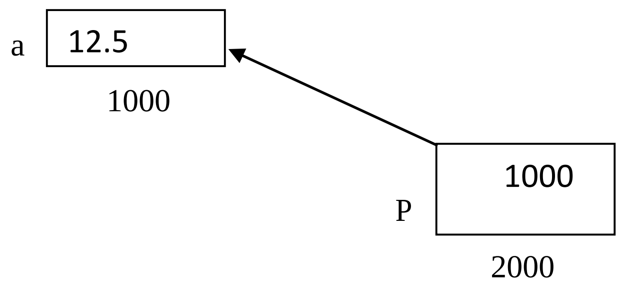
Operator	**Operator Name**	**Purpose**
*	Dereferencing Operator **OR** indirection operator	Gives Value stored at Particular address
&	Referencing Operator **OR** address of (&) operator	Gives Address of Variable

Pointer Operators

Referencing operator: (&)

& operator is a unary operator that returns the memory address of its operand. It is used to find the address of variable For example, if var is an integer variable, then &var is its address. Reference operator is also known as address of (&) operator.

```
Eg)   float a=12.5;
      float *p;
      p=&a;
```

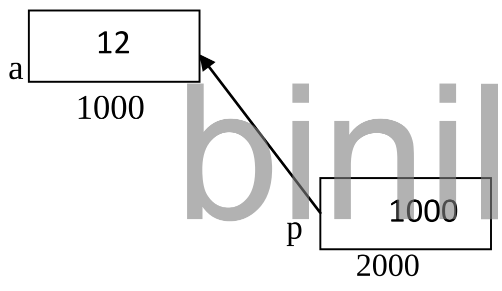


Pointer P Referencing the variable a

Dereferencing operator

The second operator is indirection Operator *, and it is the complement of &. It is a unary operator that returns the value of the variable located at the address specified by its operand. This operator is also known as indirection operator or value- at-operator

Example



```
int b;
int a=12;
int *p;
p=&a;
b=*p;     \\value pointed by p(or)value
                  at 1000=12,
so b=12
```

Pointer P Dereferencing the variable a

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int Mark = 73,
        int *Ptr;
        Ptr = &Mark;
        printf("The value of the variable Mark = %d \n", *Ptr);
        printf("The address of the variable Mark = %d \n", Ptr);
        getch();
}
```

Output:

The value of the variable Mark = 73

The address of the variable Mark = 4082

For every execution time the compiler may allot the different memory. So, the address of the variable may be different when we execute the program next time.

NULL Pointer

The pointer variable that is assigned NULL value is called Null pointer. NULL pointer is a special pointer that does not point anywhere. It does not hold the address of any object or function. It has numeric value 0

Example: for Null Pointer

```
int *ptr = 0;
        ( or )
int *ptr = NULL;
```

Use of null pointer:

- As an error value
- As a sentinel value
- To stop indirection in a recursive strucure

POINTER ARITHMETIC

Pointer is a variable that stores the address of a variable or a function.

Valid operation

- Pointer can be added with a constant
- Pointer can be subtracted with a Constant
- Pointer can be Incremented or Decremented

Not Valid operation

- Two pointers cannot be added
- Two pointers cannot be subtracted
- Two pointers cannot be multiplied
- Two pointers cannot be divided

Example:

```
 int a=10
int *p=&a;
```

p=p+1;

- The pointer holds the address 2000. This value is added with 1.
- The data type size of the constant is added with the address. p= 2000+(2*1)=2002

The following table shows the pointer arithmetic.

S.no	**Operator**	**Type of operand 1**	**Type of operand 2**	**Result type**	**Example**	**Initial value**	**Final value**	**Description**
1	Addition (+)	Pointer to type T	int	Pointer to type T				Result = initial value of ptr +int operand * sizeof (T)
	Eg.	int *	int	int *	p=p+5	p=2000	2010	2000+5*2= 2010
2	Increment (++)	Pointer to type T	-	Pointer to type T				**Post increment** Result = initial value of pointer **Pre-increment** Result = initial value of pointer + sizeof (T) Value of ptr = Value of ptr +sizeof(T)
	Eg. post increment	float*	-	float*	ftr=p++	ftr=? p=2000	ftr=2000 p=2004	
3	Subtraction -	Pointer to type T	int	Pointer to type T				Result = initial value of ptr - int operand * sizeof (T)
	E.g.	float*	int	float*	p=p-1	p=2000	1996	2000 – 1 * 4 = 2000-4=1996

4	decrement	Pointer to type T	-	Pointer to type T				**Post decrement** Result = initial value of pointer **Pre-decrement** Result = initial value of pointer – sizeof(T)
	Eg.pre decrement	float*	-	float*	ftr=--p	ftr=? p=2000	ftr=1996 p=1996	Value of ptr = Value of ptr – sizeof(T)

Pointer Arithmetic

Program : Addition of Integers with Pointer

```
#include<stdio.h>
#include<conio.h>
void main ()
{
        int a[5] = {10, 5, 20, 5, 2};
        int i, sum = 0
        for (i = 0; i < 5; i++)
                sum = sum+*(a + i);
        printf (“Total = %d”, sum);
        getch();
}
```

Output:

```
Total = 42
```

Program : Subtraction of Pointers

```
#include<stdio.h>
```

```
#include<conio.h>
void main ()
{
        double a[2], *p, *q;
        p = a;     // Assign a[0] address to p
        q = p + 1;    // Assign a[1] address to q
        printf("No. Of elements between p & q = %d", q - p);
        getch();
}
```

Output:

```
No. Of elements between p & q = 2
```