Unit
5

Functions

5.1 INTRODUCTIONS TO FUNCTIONS

A function is a module or block of program code which executes a particular task of program. Every C program has at least one function, which is main().

5.2 TYPES OF FUNCTIONS: BUILT-IN AND USER DEFINED

C functions can be classified into two categories:

1. Library functions: For example, printf(), scanf().
2. User defined functions: For example, main().

Library functions are those functions which are already defined in C library, example printf(), scanf(), strcat() etc. You just need to include appropriate header files to use these functions. These are already declared and defined in C libraries.

A **User-defined functions** on the other hand, are those functions which are defined by the user at the time of writing program.

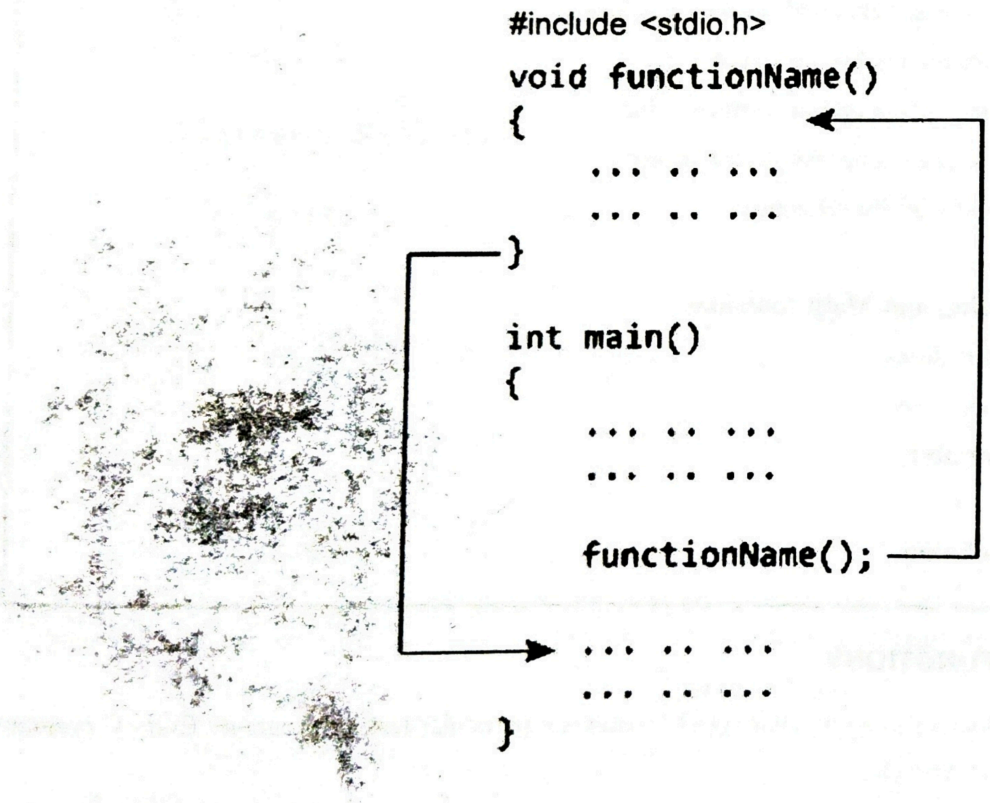
5.3 ADVANTAGES OF USING FUNCTIONS

There are several advantages of function in C or any other programming language.

- It provides modularity to your program's structure.
- It makes your code reusable. You just have to call the function by its name to use it, wherever required.
- In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.
- It makes the program more readable and easier to understand.

5.4 WORKING OF FUNCTIONS

The execution of a C program begins from the main() function.



When the compiler encounters functionName();, control of the program jumps to

```
void functionName()
```

And, the compiler starts executing the codes inside functionName().

The control of the program jumps back to the main() function once code inside the function definition is executed.

5.5 ELEMENTS OF USER-DEFINED FUNCTIONS

There are three elements of user-defined functions:

1. Function declaration
2. Function definition
3. Function call

The **Function definition**is an independent program module that is specially written to implement the requirements of the function. In order to use this function we need to invoke it at a required place in program. This is known as **Function Call**. The program (function) that is calls the function is referred to as the **Calling Program** or **Calling Function**. The calling program should declare any function that is to be used later in the program. This is known as the **Function Declaration** or **Function Prototype.**

Example:

```
int add (int,int);                    <-------  Function Definition
main ()
{
        int a=3,b=4,c;
        c = add (a,b);                <-------  Function Call
        printf("c=%d",c);
}
int add (int h, int j)
{
        int n;
        n = h + j;                    Function Definition
        return (n);
}
```

5.5.1 Function Declaration

In C, all functions must be declared, before they are invoked. A function declaration consists of four parts:

1. Function Type
2. Function Name
3. Parameter List
4. Terminating Semicolon

General format:

```
function_type function_name (parameter list);
```

In above example we have declared the function **add** as:

```
int add (int,int);
```

When function doesn't return any value then it is declared as a **void** type.

A prototype declaration may be placed in two places:

1. Above all the function(including main())
2. Inside a function definition

When we place the declaration above all the function, the prototype is referred to as a **Global Prototype**. Such declarations are available for all the functions in the program.

When we place the declaration in a function definition, the prototype is referred to as a **Local Prototype**. Such declarations are primarily used by the function containing them.

5.5.2 Function Definition (Function Implementation)

A function definition shall include following elements:

1. Function Name
2. Function Type
3. List of Parameters
4. Local Variable Declarations
5. Function Statements
6. A Return Statement

All These Six Elements Are Grouped Into Two Parts:

1. Function Header (First Three Elements)
2. Function Body (Last Three Elements.)

General format:

```
function_type  function_name (list of parameters)
{
local variable declaration;
executable statementas1;
executable statement-2;
.....
.....
return statement;
}
```

Function Name and Type:

The **Function Type** specifies the type of value that the function is expected to return to the program calling the function. If the return type is not explicitly specified, C will assume that it is as integer type. If the function is not returning anything, then we need to specify the return type as **void**.

The **Function Name** is an identifier. So, must follow the rules of identifier.

In above example the function name is **add** and declared as**int**function type because it returns integer value.

List of Parameters:

The **Parameter List** declares the variables that will receive the data sent by the calling function. They serve as an input data to the called function. The parameters are also known as arguments. In above example h and j are arguments of this function.

In above program **main()** is a calling function,**add** is a called function and **h&j** are arguments of this function.

Function Body:

The **Function Body** contains the declarations and statements necessary for performing the required task. The body is enclosed in braces, contains:Local Variable Declarations, Function Statements and A Return Statement.

If the function does not return any value, we can omit the **Return Statement**.

In above example variable **i** is local variable. **n = h + j** is an executable statements and the last one is a return statement, which returns the value of **n**.

5.5.3 Function Call

A function can be called by simply using the function name followed by a list of actual parameters (arguments), if any, enclosed in parenthesis.

In the above example we have called the function **add** by.

```
c = add (a,b);
```

Here, add (a,b) is a **Function Call**. This function call sends values of two integers to the function.

```
int add (int h, int j)
```

Which are assigned to **h** and **j** respectively. The function computes the sum of **h** and **j**, assigns the result to local variable **n**, and then returns the value **7** (because a=3, b=4) to the **main** where it is assigned to **c** again.

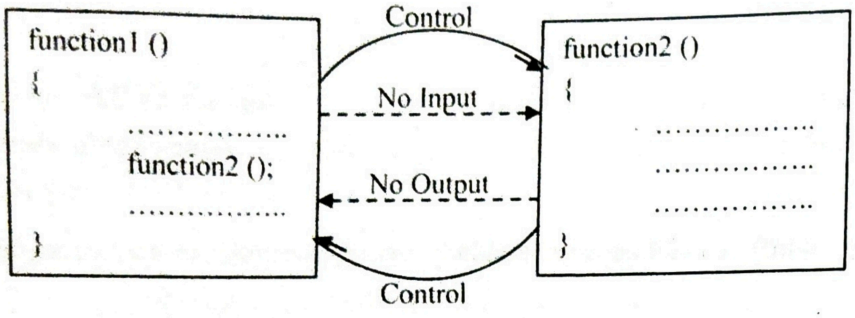
5.6 CATEGORY OF FUNCTIONS

1. Function with no arguments and no return values
2. Function with arguments but no return values
3. Function with arguments and one return values
4. Function with no arguments but return values
5. Function that return multiple values

5.6.1 Function with no Arguments and no Return Values

When the called function has no argument, it does not receive any data from the calling function. Similarly, when the called function does not return a value, the calling function does not receive any data from called function. So, there is no data transfer between calling function and called function.

As shown in above figure function1 (**main** ()) has no control over the way the **function2**receives input data.



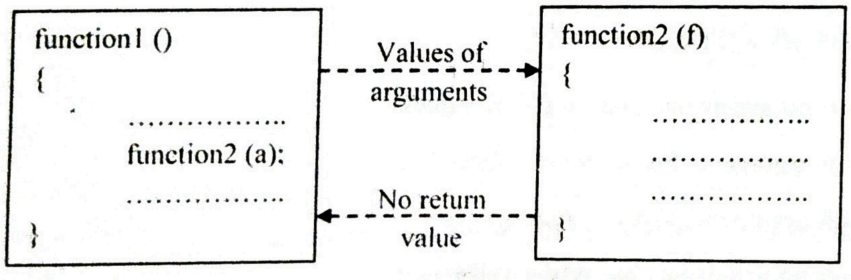
Example:

```
void add();
main()
{
add();

void add()
{
int a,b;
scanf("%d%d",&a,&b);
printf("add=%d",a+b);
}
```

5.6.2 Function with Arguments but no Return Values

The nature of data communication between the calling function and the called function with arguments but no return value is shown in below figure:



The calling function can send values of arguments to the called function using function call containing appropriate arguments. These arguments are known as **Actual Arguments**.

The called function can receive values of arguments from the calling function using function call containing appropriate arguments. These arguments are known as **Formal Arguments**.

When a function call is made, only **a copy of the values of actual argument is passed into the called function**.

For example, the function call

```
add(2,3)
```

Would send the values **2** and **3** to the called function

```
void add(int a,int b)
```

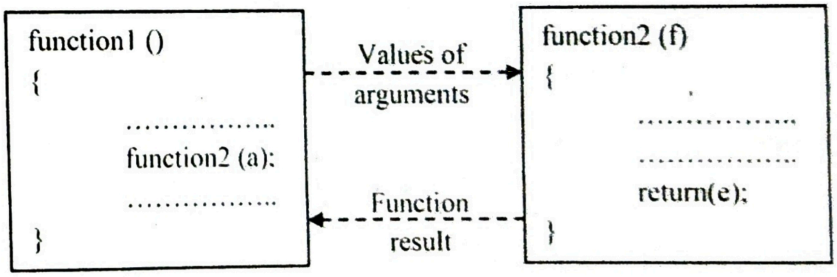
And assign **2** to **a** and **3** to **b**. The values **2** and **3** are the actual arguments, which become the values of the formal arguments inside the called function.

Example:

```
void add(int,int);
main()
{
int a,b;
scanf("%d%d",&a,&b);
add (a,b);
}
void add (int h, int j)
{
printf("add=%d", h+j);
}
```

5.6.3 Function with Arguments and one Return Values

The two-way data communication between the calling function and the called function with arguments and return value is shown in below figure:



As shown in above figure, the function call (**function2 (a)**) transfer the control along with copies of the actual arguments (**a**) to the called function(**function2 (f)**) where the formal arguments (**f**) are assigned the actual values of **a**. The called function executes line by line. When return statement executes, it passes result (value written in return statement) back to the calling function where function call is assigned.

Example:

```
int add(int, int);
main()
{
int a,b,c;
```

```
scanf("%d %d",&a,&b);
c = add (a, b);
printf("add=%d",c);
}
int add (int h, int j)
{
int n;
n = h + j;
return (n);
}
```

In above program return statement will return the value of **n** to the **main ()** function where function call **add (a,b)** is assigned to **c**. so, if **a=5** and **b=10**, then **c=15**.

5.6.4 Function with no Arguments but Return Values

The calling function does not send any arguments to the called function. But the called function passes result (value written in return statement) to the calling function by return statement.

Example:

```
int add();
main()
{
int c;
c = add ();
printf("add=%d",c);
}
int add ()
{
int a,b;
scanf("%d %d",&a,&b);
return(a+b);
}
```

In above program function call **add ()**has no argument but return statement of the called function will return the value of **n** to the **main ()** function where function call **add ()** is assigned to **c**. so, if **a=2** and **b=3**, then **c=5**.

5.6.5 Call by value and Call by reference

When function is called from another function, the arguments (parameters) can be passed in two way:

1. Call By Value (Pass By Value)
2. Call By Reference (Pass By Reference)**Or**Call By Pointer (Pass By Pointer)

In **Call By Value**, argument values are passed to the function. Means actual values of arguments are copied into the formal arguments. The called function cannot change the values of variables which are passed. Even if a function tries to change the values of passed arguments, those changes will occur in formal arguments, not in actual arguments.

Example:

```
#include <stdio.h>
void fun(int x)
{
   x = 30;
}

int main(void)
{
    int x = 20;
    fun(x);
    printf("x = %d", x);
    return 0;
}
```

Output:

```
x = 20
```

In **Call By Reference**, the reference of the arguments (address of arguments) is passed to the function. Because the address is passed, the called function can change the value of actual arguments. **Call By Reference** is used whenever we want to change the value of local variables declared in one function to be changed by other function.

Example:

```
# include <stdio.h>
void fun(int *ptr)
{
      *ptr = 30;
}

int main()
{
      int x = 20;
      fun(&x);
      printf("x = %d", x);

      return 0;
}
```

Output:

```
x = 30
```

5.7 RECURSION

Recursion means function calls itself. There must be some condition to stop recursion that is called **Termination Condition**; otherwise it will lead to infinite loop.

Example:

```
void main()
{
int fact(int);
int n,result=1;
scanf("%d",&n);
result=fact(n);
printf("Factorial = %d",result);
getch();
}
int fact(int n)
{
if(n==1)
return(1);
else
return(n*fact(n-1));
}
```

Above program returns the factorial of n. If n=3, then output will be "Factorial = 6".

Advantages of Recursion

1. Easy solution for recursively defined problems.
2. Complex programs can be easily written in less code.

Disadvantages of Recursion

1. Recursive code is difficult to understand and debug.
2. Terminating is must; otherwise it will go in infinite loop.
3. Execution speed decrease because of function call and return activity many times.

5.8 BUILT IN FUNCTIONS: STRING AND MATH FUNCTIONS

5.8.1 In-built String Functions

Function & Explanation	Example	Output

strlen
 	
Finds length of a string
	#include<stdio.h>
	#include<string.h>
	int main()
	{
	char text[5];
	int length;
	printf("Enter the string:");
	gets(text);
	length=strlen(text);
	printf("\nString Length:%d",length);
	return 0;
	}
		Enter the string:hello
		String Length:5