FUNCTIONS IN C PROGRAMMING

Every program in C has at least one function which is *main()*. Unlike Pascal or Basic, C does not provide procedures, it uses functions to cater to both requirements.

Functions are used to make a program modular and to avoid repetition of code. Any piece of code which is often used in a program is a likely candidate for being a function.

In C a function has the following structure :

<return type> function name (arguments list)
{
local variables;
code performing the necessary action
}

Let us look at a simple function which will display a message on the console.

Program 7.1

```
#include <stdio.h>

void print_message()
{
printf("Inside print_message function\n");
```

```
}
void main()
{
print_message();
printf("Back in the main program\n");
}
```

In the above program, *main()* is the entry point of the program, once program execution begins, the control is transferred to the print_message() function. Inside the function it prints the statement on the terminal and the control returns to the main routine. After the function call the program execution will continue at the point where the function call was executed.

The *printf()* and *scanf()* routines are functions too. But since these are part of the C standard library, one just have to call them. Whenever we use a *printf()* routine, execution is transferred to the *printf()* function which performs the required task, then control is returned back to the calling program.

If you do not want to return a value you must use the return type *void*, and the return statement must not be present in the body of the function (or you can just return void).

Note : If the return type of a function is omitted, then the C compiler assumes that the function will return an integer. In case a function is returning an integer value, it is a good programming practise to declare the return type instead of omitting it. This improves the readability of a program.

When a function is declared the number of arguments passed to the function and their names must also be indicated. The name chosen for an argument is called its formal parameter name. Formal parameters must be declared inside a function before they are used in the function body. One point to remember is that variables defined inside a function are known as automatic variables since they are automatically created each time the function is called and are destroyed once the function is executed. Their values are local to the function, they can be accessed only inside the function in which they are defined and not by other functions.

Program 7.2

```
#include <stdio.h>

int square(number);
main()
{
int i;
int result;
i = 10;
result = square(i);
printf ("Square of %d is %d\n", i, result);
}
int square(n)
{
int n;
int temp;
```

```
temp = n*n;
return temp;
}
```

When functions return a value to the calling routine, a *return()* statement needs to be used in the called function. Also when the function is declared we must declare the return type. In the above program the value returned by the function *square* is stored in the variable *result* in the calling program.

One can call a function from anywhere within a program. The best use of functions is to organize a program into distinct parts. The function main() can only contain calls to the various functions. The actual work of the program is performed in the functions following main().

Function prototype

In C the function prototype has to be declared before a function is used. A function prototype is information to the C compiler about the return type of a function and the parameter types that a function expects. Usually all function prototypes are declared at the start of a program.

Example:

int Min(int a, int b);

Scope of Function variables

Local variables

Local variables are local to a function. They are created everytime a function is called and destroyed on return from that function. These variables cannot be accessed outside a function.

Static variables

Static variables are declared by prefixing the keyword *static* to a variable declaration. Unlike local variables these are not destroyed on return from a function, they continue to exist and retain their value. These variables can be accessed upon re-entering a function.

Global variables

Global variables are declared outside all functions. The value stored in global variables is available to all functions within a program/application.

Recursive functions

Recursive functions are functions calling themselves repeatedly until a certain condition is met. Recursion involves two conditions. First the problem must be written in a recursive form, and second, the problem should have a loop terminating statement. If the loop terminating is missing, then the function goes into an endless loop.

A most common example of a program demonstrating recursion is calculation of factorial of an interger number.

Program 7.3

```
#include <stdio.h>

main ()
{
int num;
long int factorial(int n);  /* function prototype */
printf("Enter an integer value : ");
scanf("%d", &num);
printf("factorial of %d is %ld\n", num, factorial(num));
}
long int factorial(int n)
{
if (n <= 1)
return(1);
else
return(n * factorial(n-1));
}
```

Source : http://www.peoi.org/Courses/Coursesen/cprog/frame7.html