5. User Defined Functions

Functions:

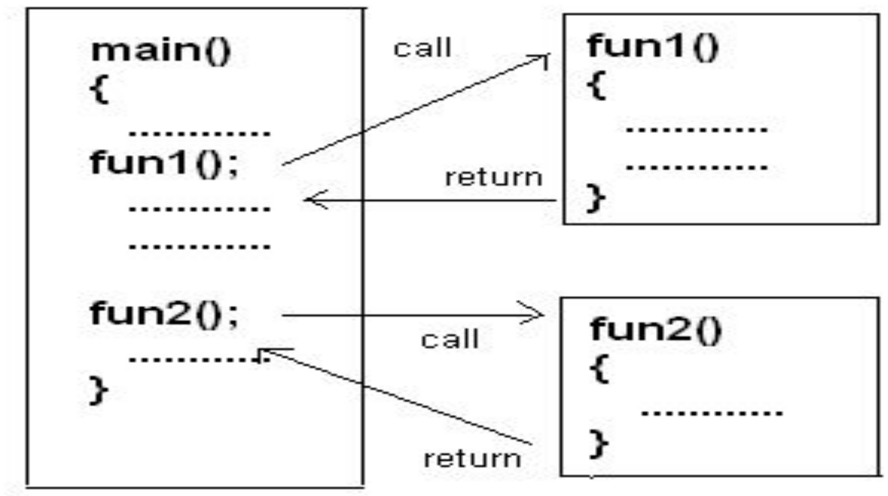
- Function is a group of statements in a single unit known by some name which is performing some well defined task.
- The functions which are created by programmer in a program are called **user-defined function**.
- While the functions which are available in library is called **library functions** or **built-in-functions.**

User defined functions:

- The function that defined by user is called User Defined functions.
  Ex- my_name ,square_area
- As the size of program increase,it becomes difficult to understand, debug and maintain it.
- User defined functions are created for doing some specific task in program.
- The code is written as one unit having name. when some portion of code is repeated in the program and when there is definite purpose of the code, we can write that part of the code as a function.
- **So the use of function reduce the size and complexity of the program.**

Library Functions:

A built in function is called library function.
Ex- printf , scanf , getchar , putchar.



Function declaration and definition

Function declaration:

syntax for declaring function is :

Function_type function_name(arguments(s));

Here, type specifies the type of value returned , function_name specifies name of user defined function, and in bracket argument(s) specifies the arguments supplied to the function as comma separated list of variables.

Ex:

Int sum(int x , int y); takes two integers and returns an integers.

Void printmessage(); takes no arguments and no return value.

Definition of Function:

It is also called function implementation.

It include following elements.

- Function name
- Function type
- List of parameters
- Local variable declarations
- Function statements
- A return statements

All the six elements are grouped into two parts, namely…

- **Function Header :** (first three elements)
- **Function Body:** (Last three elements)

A format of function are:

```
Function_type function_name(parameter_list)
{
        Local variables declaration;
        Executable statements(s);
        Return   statements;
}
```

Here type , function_name and arguments(s) have same meaning as before. The statement(s) within { } symbols indicate the body of the function. Here the actual logic of the work of function is implemented. If the return type is not mentioned, then by default it is taken as int. we can not defined function inside a function. We can call one function from other. If the function **return** type is other than void, there must be return statement in the function.

The returned value can be put in brackets but it is not compulsory.

i.e. following are valid return statements.

```
Return;           if the return type is void.
Return x;         value of x returned.
Return (x);       value of x returned
```

Program explaining declaration, definition and call of function.

```
#include<stdio.h>
void printmessage(); /* declaration */
main()
{
    printmessage(); /* call */
}
void printmessage() /* definition */
```

```
{
     Printf("Hello ! !\n");
}
```

Output:

```
Hello ! !
```

Explanation:

In above program ,

void printmessage();

The line is declaration of function printmessage,which returns no value(void), and do not require any arguments , so empty bracket(). The line is terminated by ; symbol.
The above line tells the compiler that somewhere in the program printmessage will be encountered ,which is a name of function, returning void.

The code,

```
void   printmessage()
{
      Printf("Hello ! !\n ");
}
```

Is the definition of function. In the body of function only one statement printf() is written. When the above function is called, **Hello ! !** message will be displayed on screen.

NOTE : definition of function is outside the main() function. There is no semicolon(;) after } symbol.

The line in main() function,

Printmessage();

Is call to the printmessage() function, so the statements in the body of function will be executed.

In above program, declaration of printmessage() function is needed. It is not necessary to write the declaration in all programs.

The above program can be rewritten without using declaration of the function as shown below.

```
#include<stdio.h>
/* declaration   of function not needed */
void printmessage()            /* definition */
{
      Printf("Hello ! !\n");
}
main()
{
     printmessage();                          /* call */
}
```

Output:

```
        Hello ! !
```

Explanation:

In above program, the only change is that the definition of function is written before main() function. Because of this, declaration is not required. New sequence is definition first and then call of function in main().

NOTE: In the program, if the definition come first ,and then the call to the function is made, then declaration is not needed.

Formal and Actual parameters.

```
void sum(int a, int b);
```

is declaration of sum function. Here, **a and b are formal parameters. Formal parameters are variables which are used in the body of the function and in prototype.**

Formal parameter: Parameters used in prototypes and function definitions are called formal parameter.

Actual parameter: Parameters used in function call are called actual parameters.
The formal and actual parameters must match exactly in type order and number.

Scope of variables :

By the scope of a variable, we mean in what part of the program the variable is accessible is dependent on where the variable is declared. There are two types of scope for a variable- **Local** and **Global.**

Local variables:

The variables which are declared inside the body of any function are called as local variables f or that function.

These variables can not be accessed outside the function in which they are declared.

Global variables:

The variables which are declared outside any function definition is called as global variable.

These variables are accessible by all the functions in that program.

i.e. All the functions in program can shared global variable.

Local variables	**Global variables**
Declared inside function body	Declared outside function body
Not initialized automatically	Initialized automatically by value 0
Use of local variables advisable	Too much use of global variables make program difficult to debug, so use with care

Parameter passing to function:

Where a function is called from other function , the parameter can be passed in two ways.

Call by value

In call by value , argument values are passed to the function, the contents of actual parameters are copied into the formal parameters.

The called function can not change the value of the variables which are passed.

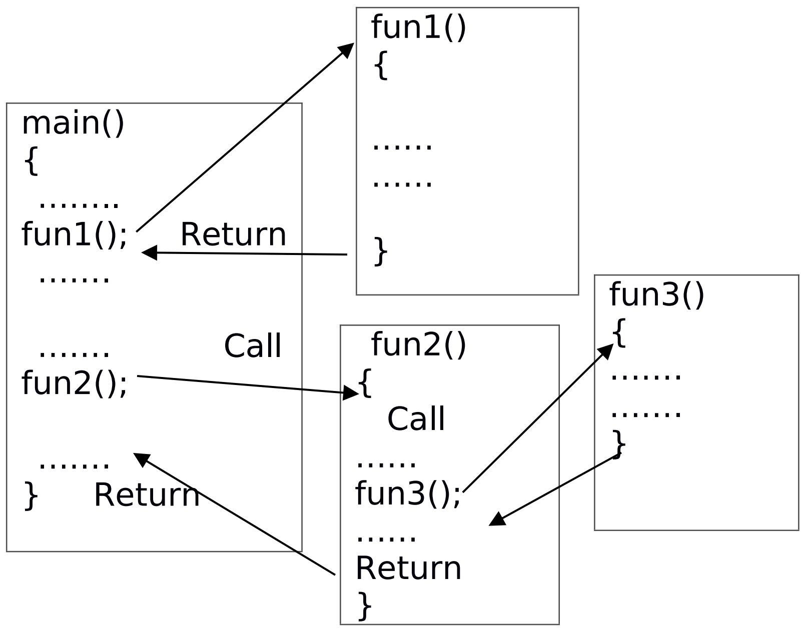
Call by reference

In call by reference, as the name suggests, the reference of the parameter i.e. address(pointer) is passed to the function and not the value.

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.

Nesting of functions:

As mentioned earlier, we can have many user defined functions in a program.



In figure main() function calls fun2() function which internally calls another function fun3(). When fun3() is over, control returns to the function which is called it.

Function fun2(), when fun2() is over control returns back to the main() function.

In above program , the line

void printmessage();

is declaration of function printmessage,which returns no value(void), and do not require any arguments , so empty bracket().

The line is terminated by ; symbol. The above line tells the compiler that somewhere in the program printmessage will be encountered ,which is a name of function , returning void.

Recursion:

recursion is the process by which a function calls itself.

Example :

```
 Void main()
{
        Int i=5;
        printf(“%d”,i);
        main();
}
```

Thus, recursion is the process by which a function calls itself . Because the function calls itself ,there must be some condition to stop recursion; otherwise it will lead to infinite loop.

Advantages:

- Easy solution for recursively defined problems.
- Complex programs can be easily written with less code.

Disadvantage:

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

Function with array as parameter:

The array can be passed to function as argument as like individual variables like a char, an Integer,a double etc.

Example:

```
z=max(a,n);
```

In this function call, **a** is an array of **n** integers.

The array **a** is passed as first argument while the size of the array is passed as second argument to the function max_n0 is written as flows.

The max() is written as:

```
int max(int v[], int size)
  {
        int i,max;
        max=v[0];
        for(i=0;i<size;i++)
              {
                    if(v[i]>max)
                    max=v[i];
                }
        return(max);
 }
```

In above function, the array variable v is not given the size because its Size is the size of the actual argument (here array a)passed in function call. When array is passed to the function,actually array elements are not passed,but only the array name which is the address of the first element in array is passed.

Storage classes:

The storage classes of variable determines the scope and lifetime of a variable.

The variable values can be stored in computer memory or in registers of CPU.

There are 4-storage classes:

1) Auto
2) Register

3) Static
4) Extern

Syntax:

storage_specifiers data_type variable_name;

1)Automatic variable

- It is declare inside the function.
- They are created when function is called and destroyed automatically when the function exited.
- It is referred local or internal variable.
- We can define automatic variable using 'auto' keyword.

2)Register variable

- A value of variable should be kept into register, instead of memory.
- Register access is much faster than memory access.
- We can define register variable using '**register**' keyword.

3) **Static variable**

- The value of variable persists until the end of program.
- A static variable is initialized only once, when the program is complied.
- We can define static variable using '**static**' keyword.

Internal static variable:

- The value of variable extends up to the end of function.

External static variable:

- External static variable declared outside of all functions and is variable to all the functions in the program.

5) **External variable:**

- External variable declared outside the function.
- It is refers to as a global variable.
- Global variable can be accessed by any function in the program.
- We can define extern variable using ‘**extern**’ keyword.

Preprocessor:

The C Preprocessor is a macro Preprocessor (allows you to define Macros) that transforms your program before it is compiled.these transformations can be inclusion of header file, macro expansions etc.

All pre processing directives begins with a # symbol.

Example:

```
#define PI 3.14
```

Directive	**Function**
#define	Defines a macro substitution
#undef	Undefines a macro
#include	Specifies the files to be included
#ifdef	Test for a macro definition
#endif	Specifies the end if.
#ifndef	Tests whether a macro is not defined.
#if	Test a compile-time condition.
#else	Specifies alternatives when if test fails

Macro substitution:

Macro substitution has a name and replacement text, defined with #define directive. The Preprocessor simply replaces the name of macro with replacement text from the place where the macro is defined in the source code.

Syntax:

```
#define identifier string
```

Example:

```
#define pie 3.14
#define count 100
#define capital “Delhi”
```

File inclusion :

File inclusive Directories are used to include user define header file inside C Program.File inclusive directory checks included header file inside same directory (if path is not mentioned).File inclusive directives begins with **#include.**

Syntax:

```
#include<filename>

#include “filename”
```

Example:

```
#include<stdio.h>
#include<string.h>
#include”test.c”
```