Unit 4

Functions & Pointers

Functions:

→ A Function is a group of statements that together perform a task.

→ The program may become too large & complex and as a result the task of debugging, testing and maintaining becomes difficult.

→ If a program is into functional parts then each part may be independently coded & later combined into a single unit.

→ These independently coded programs are called Subprograms, that are much easier to understand, debug & test.

Types of Function:

1. Library Function
2. User-defined function

Library Functions:

→ Declared in the C header files.

scanf(), printf(), gets(), puts(), ceil(), floor(), etc.

User defined Functions:

which are created by the C programmer.

That can be used many times.

→ It Reduces the Complexity of a large program & optimizes the code.

(main, user creates own name).

→ The Fun. are classified as one of the derived data types in C.

→ In order to make use of a user-defined Functions.

There elements:

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

Function definition:

It is an independent program module that is specially written to implement the requirements of the function.

Function call:

In order to use this function we need to invoke it at a required place in the program.

This known as the "Function call".

It is refers to calling program (or) calling function.

Function Declaration: (or) Function prototype

→ The calling program should declare any fun. that is to be used later in the pgm. This is known as the "Function declaration (or) prototype".

function definition:

A Function definition consist of a "function header & a function body":

return_data_type Function_name (data_type variable1, parameter list

datatypevariable2, ...)

↓

Fun. header

{

statements :

Return (Variable) ; → Function body.

}

which is includes: element (in string)

1. Function name
2. Function type
3. list of parameters → Fun. header.
4. logical variables declarations.
5. Function statement.
6. Return statement

Fun. body.

Fun. header:

Function name & Function types:

→ Actual name of the fun. The fun. name and the parameter list together constitute the function signature (add, sub, mul, dive)

type:- A fun. may or may not return a value. If is the data type of the value the fun. returns. Some functions perform the desired operation without returning a value. In this case, the return data type.

Parameter list:

→ Declare the variables that will receive the data sent by the calling program.

→ They save the I/P data to the function to carry out the specified task.

→ Since they represent actual i/p values are often refered to as formal parameters.

→ It can also be used to send values to the calling program.

Function_type function_name (parameter list) → header statement

{

Logical variable declaration;

Function statement;

Return statement;

}

→ header statement (no. semicolon)

Function Body:

→ It contains the declaration and statement necessary for performing the required task.

→ The body enclosed in braces contain 3-parts:

1. local declaration :→ That specify the variables needed by the function.
2. Function statements :→ That perform the task of the function.
3. Return Statement :→ That returns the values evaluated by the fun.

For example:

```
Float mul (float x, float y)
{   float result;      - /* local variable */
    result = x * y;    /* compute the product */
    return (Results);  /* return the result */
}
```

Function call:

→ we create main fun to call parameter

Simply using the fun. name followed by a list of actual parameter (or arguments) if any enclosed in parenthese

eg:

```
main()
{ int y;
  y = mul (10,5) ; /* Fun. call */
  Printf ("%d", y);
}
```

Fun. name
parameter list
(all.

```
int mul (int x, int y)
{
  int p ;          /* local variable */
  p=x*y ;          /* x=10, y=5 */
  return(p) ;
}
```

o/p : 50

Function Declaration :-

It consist of 4-types

1. Function type (return type)
2. Function name
3. parameter list
4. Terminating semicolon

Syntax:

```
Fun_type Fun_name (parameter list) ;
```

Int (or) float add, sub, mul (int x, y) ;

Sample program:

o/p

50

```
#include <stdio.h>
#include <conio.h>
Mul (int x , int y) ;   /* Fun. Declaration */
Void main()
{
  int y ;
  y = Mul(10,5) ;       /* Fun. call */
}
int mul (int x , int y ) -> /* Fun. definition */
{
  int p ;               /* local variable */
  p=x*y ;               /* Fun. Statement */
  return (p);           /* Return statement */
}
```