UNIT - III

15/6/17

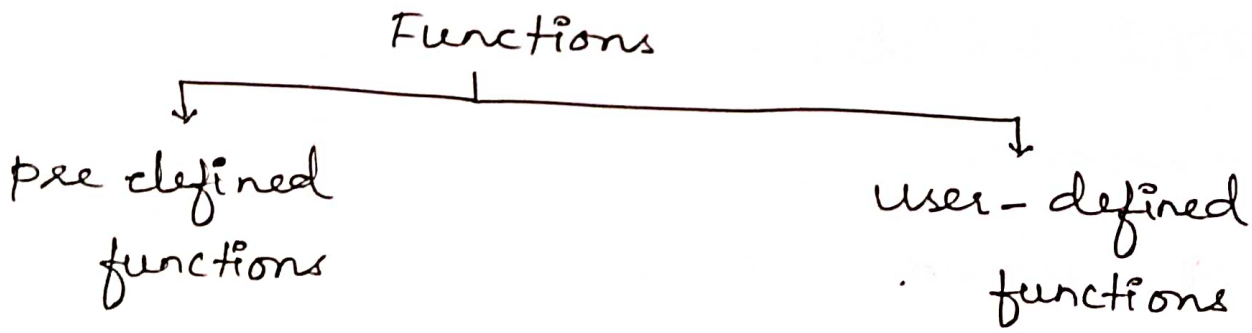
FUNCTIONS AND POINTERS.

INTRODUCTION :-

* we have already seen that one of the main advantage in 'c' language is that 'c' function are easy to define and can be used.

* A function is a set of instructions that performs a specified task which repeatedly occurs in the main program.

* Functions are classified into two type as shown below



* The difference between these two categories of functions is that library functions are not required to be written by the programmer, where as a user defined function has to be written by the programmer at the time of programming.

PREDEFINED FUNCTIONS: [LIBRARY FUNCTIONS]

* C language provides number of functions that are used to carryout various commonly used operations or calculations called library functions.

* C language provides, library functions for the following tasks.

* The library functions, that are used to carryout read/write operations.

* Functions that are used to carryout certain mathematical operations.

* Functions that are used to perform operations on files. and also to perform operations on Strings and characters.

* Functions that are used to control C language etc..,

* Some of the library functions return a data item to their accesspoint.

* Some of the functions returns either True or false (1 or 0) to their access point depending on

the condition.

* Some of the functions carryout specific opn. on data and do not return anything.

* A library function can be accessed by simply writing the function name, followed by a list of arguments, that are used to specify some information being passed to the function.

* The arguments that are passed to the function must be enclosed with in parentheses and separated by commas.

* The arguments may be constants, Variables, or expressions, if there no argument to the function the parentheses is must.

* The library functions are usually grouped together as a programs in separate library files. There are discussed in detail below.

EXAMPLE:-

Conversion of lower case to uppercase

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
  char txt[80];
  int i, t;
  clrscr();
  for(i=0; (txt[i]=getchar())!='\n'; ++i)
  t=i;
  for(i=0; i<t; ++i)
  putchar(toupper(txt[i]));
}
```

Hello, hi

HELLO, HI

Trigonometric function

functions	Meaning	Argument	Value
sin(x)	sin x	float in radian measure	float
cos(x)	cos x	"	"
tan(x)	tan x	"	"

Character functions.

functions	Meaning	Argument	Value
toascii(x)	returns the integer value for the particular character	character integer	integer
tolower(x)	to convert the character to lowercase	character integer	character
toupper(x)	to convert the character to upper case	character	character

USER DEFINED FUNCTIONS:

* These are written by the Programmer to perform a particular task, that is repeatedly used in main program.

* These functions are helpful to break down a large program into a number of smaller functions.

NEED FOR USER-DEFINED:

* While it is possible to write any pgm using the main() function and it leads to a number of problems, such as

i) The program becomes too large and complex

ii) The task of debugging, testing and maintenance becomes difficult.

* If a program is divided into parts, then each part may be independently coded and later combined into a single program.

* These sub-programs are called functions. and much easier to understant, debug and test.

* There is a situation, when some type of operations or calculations are repeated at many places throughout the program.

* In such a case, we may repeat the program statements, wherever they are needed.

* Another approach is to design a function that can be called and used, whenever required it saves both time and memory.

ADVANTAGES:-

* The length of the source program can be reduced by dividing it into the functions.

* Easy to locate and debug an error

* the user-defined function can be used in many other source programs, whenever necessary

* functions avoid coding of repeated programming of the similar instructions.

* Functions enable a programmer to build a customised library of repeatedly used routines.

* functions facilitate top-down programming

ELEMENTS OF USER_DEFINED FUNCTIONS.

In order to write an efficient user defined functions, the Programmer must familier with the following three elements

a) function definition

b) function call

c) function declaration.

a) function definition

It is the process of specifying and establishing the user defined function by specifying all of its elements and characteristics.

b) function declaration.

Like the normal variables in a program the functions can also be declared before they defined and invoked.

SYNTAX:-

returntype function name (Parameter list);

Where

returntype function → is the returntyp of function

function_name → is the name of the function

parameter list → are the list of parameters that the function can convey.

Eg:- int add (int x, int y, int z);

While declaring a function, follow the points given below

i) the list of parameter must be separated by comma.

ii) the names of the parameter is optional but datatype is must.

iii) If the function does not return any value then the return type void is must.

iv) If there is no parameters, simply place void in braces.

v) The parameters datatype must match the parameter datatype in function definition.

SYNTAX:-

```
datatype function_Name (parameter list);
parameters declaration;
{
local Variables declaration;
.....
body of the function;
.....
return (expression);
}
```

where

datatype → The type of data that the function is going to return to its main program

function_name → The name of the function

parameter list → These are the list of parameters that are transferred to the function from the main program. These are called the arguments. There list must be seprated by commas and has no termination after the parenthesis.

PARAMETERS :-

parameters provides the data communicatio between the calling function and called function

There are two types of parameter are there.

i) Actual parameters

ii) formal parameters

i) Actual parameter :

There are the parameters transferred from the calling program (main program) to the called program (function).

ii) formal parameter

These are the parameters transferred into the calling function (main program) from the called program (function)

Example :-

```
main()
{
 ....
 .....
 fun1(a,b);
 ...
}....
```

```
fun1(x,y)
{
 ....
 ....
 ....
}
```

Where a,b are the Actual parameters.

x,y are the formal Parameters.

All the prog parameters are declared is parameters declaration.

i) If there is less actual parameters than the formal parameter, the unmatched formed parameters will hold the garbage values.

ii) If there is more actual parameters than the formal parameter, the unmatched formed extra actual parameter will be discarded

iii) If there is any mismatch of datatypes they also hold the garbage values.

THE RETURN STATEMENT:-

The return statement may or may not send back any values to the main Program (calling pgm). If it does, it can be done using the return statement

SYNTAX:- return; or

return (exp)

Where a,b are the Actual parameters.

x,y are the formal Parameters.

All the prog parameters are declared is parameters declaration.

i) If there is ~~more~~ less actual parameters than the formal parameter, the unmatched formed parameters will hold the garbage values.

ii) If there is more actual parameters than the formal parameter, the ~~unmatched~~ formed extra actual parameter will be discarded

iii) If there is any mismatch of datatypes they also hold the garbage values.

THE RETURN STATEMENT:-

The return statement may or may not send back any values to the main Program (calling pgm) If it does, it can be done using the return statement

SYNTAX:- return; Or

return (exp)

where

return; does not return any values.

return(exp); returns the specified exp value to main program.

A function can contain more than one return statements, when the return value can be returned based on certain condition.

EXAMPLE:-

```
if (a>0)
    return(1);
else
    return(0);
```

> By default all functions return int datatype.

If a function can return some other than int type, then we must specify the datatype to be return.

Some important points to be noted using return statement.

* The return statement can return only one value from the called function to the calling fxn.

* The return statement can be present any where in the function.

* The return statement not necessary at the end of the function.

* If the called function does not return any value, then the keyword void must be used as the return type specifies.

* Number of return statements used is a function are not restricted.

* parenthesis used around the expression in a return statement is optional

FUNCTION CALLING:-

A function can be called by typing the function name in a source pgm with parameters, if presence with in parentheses.

EXAMPLE:- function call without parameters

```
main()
{
  message();
  printf("Main message");
}

message()
{
  printf("function message\n");
}
```

OUTPUT:-

Function Message

Main Message.

When we call the function message() the control passes to the function message(). the activity of main() is temporarily suspended.

the message() function runs and the Statements are executed and the control returns next line from where the function is called in main ppgm

main() function becomes calling fxn
message() function becomes called fxn

EXAMPLE:-

function call with parameter

```
main()
{
  int
  x= add(3,9);    /* fxn call */
  printf("Results... %d", c);
}
```

The compiler automatically transferred the control to the function add(), then the function add() is executed after the add() function returns some value to the calling fxn

```
main()
{
int x;        /* fxn call */
 c= add(3,9);
printf("Results %d %c);
}
```

```
int add-(int a, int b)
{
 int x;
 x = a+b;
 return (x);
}
```

There may be various ways to call a function.

Eg:- add (3, 9);

add (k, 9);

add (3, 1);

add (l + k, 9);

FUNCTION PROTOTYPES:-

The functions are classified into the following types depending on whether the arguments are present or not whether a value is returned or not.

These are also called function prototypes.

i) function with no arguments and no return value

ii) function with arguments and no return value

iii) function with arguments and with return value

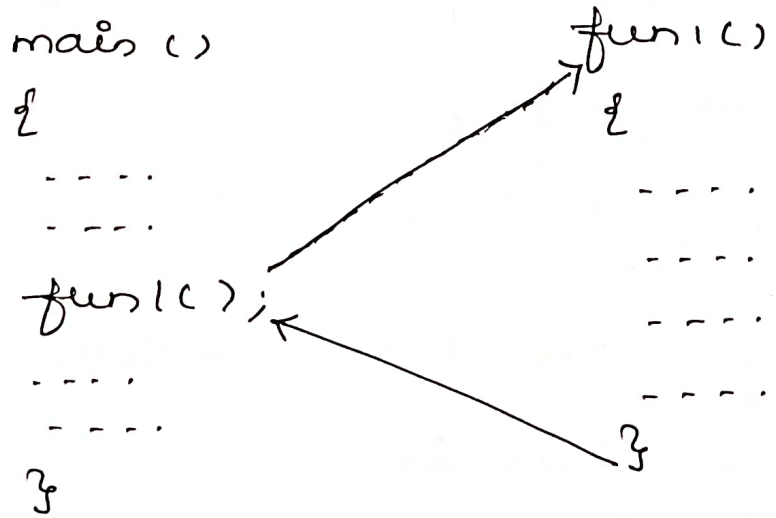
iv) function with no arguments and with return value.

i) function with no arguments and no return values

In these functions there is no data transfer takes place between the calling function and the called function

i.e., the called program does not receive any data from the calling program and does not send back any value to calling program.

SYNTAX:-



The dotted lines indicates that, there is only transfer of control but no data transfer.

EXAMPLE :- Addition of two numbers

```
main()
{
void add(void);
add;
}
void add()
{
int a, b, c;
printf("Enter two nos....");
scanf("%d %d", &a, &b);
c= a+b;
printf("sum is.... %d", c);
}
```

OUTPUT :-

Enter two numbers.... 10 20

Sum is .... 30.

FUNCTION WITH ARGUMENTS AND NO RETURN VALUES

In this function the data are transferred from calling function to called function.

ie., The called program receives some data from the calling program and does not send back any values to calling program (one way communication)

SYNTAX:-

```
main()
{
 ....
 ....
 fun1(a,b);
 ....
 ....
}
```

```
fun1(x,y)
{
 .....
 ....
 .....
}
```

The condition the continuous line indicate data transfer and dotted line indicates transfer of control

EXAMPLE: Addition of two numbers.

```
main()
{
  void add(int, int);
  int a,b,c;
  printf("Enter two values: ");
```

```
    Scanf ("%d %d", &a, &b);
    add(a,b);
}
void add (int x, int y)
{
    int z;
    z= x+y;
    Printf("Sum is .... %d", z);
}
```

OUTPUT:-

Enter two values : 10 20

Sum is 30

iii) function with arguments and with return value

In this function the data is transferred between the calling function and called function

i.e., The called function receives some data from the calling program and send back a value return to the calling program. (Two way connection.)

EXAMPLE: Addition of two numbers

```
main()
{
int add (int, int);
int a, b, c;
printf(" Enter two numbers: ");
scanf("%d %d", &a, &b);
c = add (a, b);
printf("Result is ... %d", c);
}

int add (int x, int y)
{
  int z;
  z = x+y;
  return (z);
}.
```

OUTPUT:-

Enter two numbers 10 20

Result is ... 30.

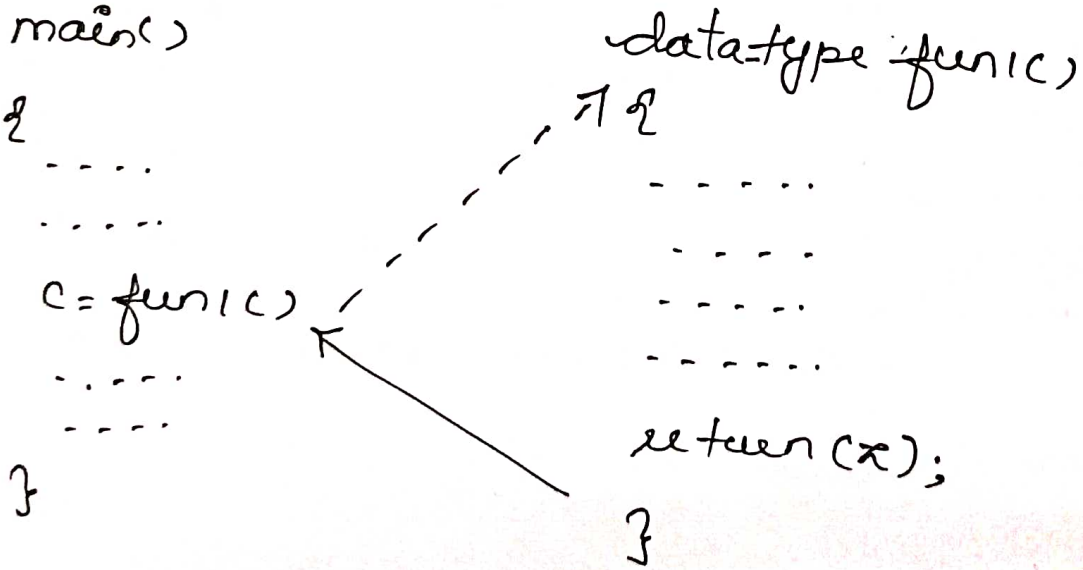
In the above program, the function add is calling program transfers some parameters (a,b) to called program and the called program send back a value (z) to calling program.

iv) FUNCTION WITH NO ARGUMENTS & WITH RETURN VALUE

In this function, there is one way data communication takes place

i the calling program cannot pass any arguments to the called program but, the called program may send some return value to the calling program.

SYNTAX:-



EXAMPLE:- Addition of two numbers:-

```
main()
{
  int add();
  c= add();
  print("Results in... %d", c);
}
  int add()
  {
    int a, b, c;
    printf("Enter two numbers:");
    scanf("%d %d", &a, &b);
    c= a+b;
    return(c);
  }
```

OUTPUT:-

```
Enter two numbers:- 10, 20
Result is .... 30.
```

CALL BY VALUE & CALL BY REFERENCE:-

. In 'C' language there are two ways, that the parameters can be passed to a function, they are

i) call by value

ii) call by reference.

i) CALL BY VALUE:-

This method copies the values of actual parameters into the formal parameters of the functio

Here, the changes of the formal parameter cannot affect the actual parameters.

Eg:- find the cube of given value ?.

```
#include<stdio.h>
int cube(int x)
{
 main()
 {
  int n=5;
  printf("cube of %d is.... %d", n,
                                cube(n))
 }
```

```
int cube(int);
{
    x = x*x*x;
    return(x)
}
```

OUTPUT:- cube of 5 is.:. 125

In this example, the value of argument to the function cube(), 5 is copied to the parameter 'x'. The expression x = x*x*x is evaluated, and only the local variable 'x' is modified.

) CALL BY REFERENCE:-

It is another way of passing parameters to a function. Here the address of arguments are copied into the parameter inside the function, the address is used to access the actual arguments used in the call.

Here pointers are passed to function, just like any other arguments. Here we need not declare the parameters as pointers type.

Eg:- Interchanging two values.

```
#include <stdio.h>
void interchange(int *a, int *b);
main()
{
  int i=5; j=10;
  printf("i & j values before %d %d \n",i,j);
  interchange(&i, &j);   /* pass address to function*/
printf("i & j values after:
        %d %d \n",i,j);
}
```