UNIT 5

USER DEFINED FUNCTIONS

Page No.
Date :

- It is a block of code / Group of statements / self contained block of statement / basic building blocks in a program that performs a particular task.

- It is also known as procedure or subroutine or module in other programming language.
- To perform any task, we can create function. A function can be called many times.
- It make's the code optimized.
- It reduces complexity of big program.
- It is created by user.

* ELEMENTS OF USER DEFINED FUNCTIONS :-
1. Function Declaration
2. Function call.
3. Function Definition

* 1. FUNCTION DECLARATION :-
- It is process that tells compiler about function name.
- SYNTAX :-
  return_type function_name (parameter/argument);
  ~~[illegible]~~ return-type function_name ( );

- Eg:- int add (int a, int b);
  int add ();

2. CALLING A FUNCTION :
- When we call any function, control goes to function body & execute entire code.
- SYNTAX : function_name ( );
  function_name (parameter/argument) ;
  return value / variable = func^{n}_name (parameter/argument).

- Eg:- add ();
  add (a, b);
  c = fun (a, b);

2 DEFINING A FUNCTION :-
- It means writing Logic inside function body
- SYNTAX :-

```
return_type. function name (parameter list) // Func^n Header
{
      declaration of variables;
      body of function;
      return statement; // expression or value.
                            It is optional
}
```

Eg-

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

Eg.

```
#include <stdio.h>
#include <conio.h>

Void Sum (); // Declaring a function.
Clrscr ();
int a=10, b = 20, c;

Void Sum () // Defining a function.
{     c = a+b ;
```

Page No.
Date :

```
printf ("sum : %d", c);
}
void main ()
{
    sum () ;    //calling the function
}
```

OUTPUT

```
sum : 30
```

* CATEGORY OF FUNCTIONS.

1. FUNC^N WITH NO PARAMETERS & NO RETURN VALUE:

- No data transfer bet^n calling func^n & called func^n.
- There is flow of control from calling to called func^n
- When no parameters are there, the func^n cannot receive any value from calling func^n.

```
# include <stdio.h>
void sum ()
{
    int x, y ;
    printf (" Enter x & y \n");
    scanf ("%d %d", &x, &y);

    printf ("Sum of %d & %d is : %d", x, y,
                x+y) ;

}
int main ()
{
    sum () ;
    return 0;
}
```

OUTPUT :

```
Enter x & y
Sum of 20 & 0 is : 20
```

- In the above program function sum does not take any arguments & has no return values. It takes x & y as inputs from the user & prints them inside void function.

2] FUNCTION WITH NO ARGUMENTS & WITH RETURN VALUE :

- Functions that have no arguments but have some return values. Such functions are used to perform specific operations & return their value.

```
# include <stdio.h>
int sum ()
{
    int x, y, s=0;
    printf ("Enter x & y \n");

    scanf ("%d %d", &x, &y);

    s = x+y ;

    return s;
}
```

```
int main ( )
{
  printf ("Sum of x & y is %d", Sum ( ));

  return 0;
}
```

OUTPUT

```
Enter x & y
Sum of x & y is 20
```

In the above program function sum does-not take any arguments & has a return value as an integer type. It takes x & y as inputs from user & returns them.

3] FUNCTION WITH ARGUMENTS & NO RETURN VALUE:

- Functions that have arguments but no return values such functions are used to display or perform some operations on given arguments.

```
# include <stdio.h>
 void sum (int x, int y)
 {
     printf ("sum of %d & %d is : %d",
               x, y, x+y );

 }
 int main ()
 {   int x, y ;
     printf ( "Enter x & y \n") ;
scanf ("%d %d", &x, &y);

 Sum (x, y);   // function call.

 return 0;

}
```

- OUTPUT

```
Enter x & y
Sum of 0 & 0 is : 0
```

- In the above program, function sum takes x & y as arguments & has no return value. The main function takes x & y as inputs from the user & calls the sum function to perform print operation on the given arguments.

4] FUNCTION WITH ARGUMENTS & WITH RETURN VALUE :-

- These functions are used to perform specific operations on given arguments & return their values to the user.

```
# include <stdio.h>
 int sum (int x, int y)
 {
    return x+y;
  }

 int main ()
{
     int x, y;
printf (" Enter x & y \n");
scanf (" %d %d", &x, &y);

printf ("Sum of %d & %d is: %d",
        x, y, sum (x,y));

 return 0;

}
```

- OUTPUT

```
Enter x & y
sum of 0 & 0 is: 0
```

- In the above program, func^n sum takes 2 arguments as x & y and has return value as an integer type.
- The main function takes input x & y from the user & calls the sum function to perform a specific operation on given arguments & returns the value.

★ STRUCTURE :-
- It is user defined data type which holds different data type in a single variable.
- It is combination of primitive & derived data type.
- Variable inside the structure are called members of structure.
- Each element of structure is called member.
- 'Struct' keyword is used to define a structure.
- SYNTAX :-

```
    Struct structure_name /tag name.
```

```
{ data-type member 1;
  data-type member 2;
     :
  data-type member n;
};
```

Eg:-
```
struct employee
{ int id ;
  char name [50];
  float salary ;
};
```

- Here, 'struct' is the keyword 'employee' is the tag name of structure, id, name & salary are members of structure.

* SYNTAX TO CREATE STRUCTURE VARIABLE :-
struct tagname/structure_name variable ;

* DECLARING STRUCTURE VARIABLE :-
1. By 'struct' keyword within main function.
Declaring structure variables separately

Eg:-
```
struct employee
{ int id ;
  char name [50] ;
  float salary ;
};
struct employee e1, e2 ;
```

- If no. of variables are not fixed use this approach. It provides flexibility to declare structure variable many times.

2. Declaring variable at the time of defining structure.

Eg:-
```
Struct employee
{ int id;
  char name [50];
  float salary;
} e1 ,e2 ;
```
- If no. of variables are fixed use this approach. It saves your code to declare variable in main() function.

* STRUCTURE INITIALISATION:-

- It can be initialised at compile time

Eg:-
```
Struct patient P1 = {180.75, 73, 23};
            OR
Struct patient P1;
P1.height = 180.75;
P1.weight = 73;
P1.age = 23;
```

* ACCESSING STRUCTURE / MEMBERS OF STRUCTURE.

1. Member or dot operator [.] :- used when variable is of normal type.
2. Structure pointer operator [->] :- used when variable is of pointer type.

Eg:-
```
Struct book
{ Char name [20];
  Char author [20];
  int pages;
};
Struct book b1;
```

For accessing the structure members from above example :-

b1.name , b1.author, b1.pages :

* PROGRAM BASED ON STRUCTURE :-

```
# include <stdio.h>
# include <Conio.h>
 struct emp
 { int id;
   char name [36];
   float sal;
  };
 Void main ( )
 {   struct emp e ;
     clrscr ( );
   printf ("Enter employee Id, Name, salary:");
   scanf ("%d", &e.id);
   scanf ("%s", &e.name);
   scanf ("%f", &e.sal);
   printf ("Id : %d", e.id);
   printf ("\n Name : %~~d~~s", e.name);
   printf ("\n salary : %f", e.sal);
   getch ();
   }
```

OUTPUT :-

```
Enter employee Id , Name , salary : 5 spidy 45000
 Id : 05
 Name : spidy
 salary : 45000 .
```