UNIT- 5

FUNCTIONS

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

Types:

- standard library functions
- User defined functions

Standard library functions:

- built-in functions
- Eg mathematical operations, I/O processing, string handling
- these functions are available for use by including the header files
- Eg) printf(), scanf() using #include<stdio.h>
  pow(), sqrt() using #include<math.h>

User defined functions:

- Users can define functions
- based on complexity and requirement of the program, users can define the function

How function works:

```
#include<stdio.h>
void functionname()          // functiondefinition
{ ...
}
int main()
{ ...
  functionname();            // functioncall
  ...
  ...
}
```

* main() is the function that will be executed.

* When the function call is present the control goes to the function definition and the definition statements are executed.

* Then returns backs to the main().

Syntax of function definition:

```
returntype fnname(parameterlist) //fn header
{
    body of the function
}
```

The function header (fn. header) has 3 parts.

* Returntype: the datatype of the value the function returns. If only some operations are done and no value is returned, then void is the return type.

* functionname: Actual name of function

* parametertype-type, order and number of arguments required

If no parameters then empty ()

function name & parameter list together forms the function signature

Function declaration: Tells the compiler the function name and how to call the function. Actual body of function can be done separately.

Syntax:

returntype functionname(parameterlist);

In declaration, name of parameter is not compulsory.

Function call:

To use a function, the function has to be called. Calling is done using function name and the argument values (parameter value). When function call, evaluated, control goes to function definition.

Example:

```
#include <stdio.h>
int max(int num1, int num2); //fn declaration
int main()
{
    int a=100;
    int b=200;
    int ret;
    ret = max(a,b);      // fn call
    printf("Max value is %d", ret);
    return 0;
}

int max(int num1,int num2) // fn definition
{   int result;          // local variable
    if(num1>num2) result=num1;
    else result=num2; return result;
}
```

output:
Max value is 200

Function Arguments:
If a function uses arguments, it must declare variables that accept the values of the arguments. These variables are called formal variables.
Formal variables are like local variables (variables inside the function) and are created upon entry to the function and destroyed upon exit.

* Call types: (In C, default call by value)
i) Call by value:
* Method copies the actual value of an argument into formal parameter of the function.
* Changes made to the parameter inside the function have no effect on the argument.
ii) Call by reference:
* Method copies address of an argument into the formal parameter. Inside the function, address is used to access the actual argument used in the call.
* Changes made to the parameter affect the argument.

* Function argument types.
i) with argument:
= declared and defined with

parameter list.
- values of parameter passed during function call
- Eg int sum(int x, int y); // fn declaration
sum(10, 20); // fn call

ii) without argument:
- No parameter list
- No value passed during function call
- Eg int show(); // declaration
show(); // call

i) Call by Value: Eg

```
#include<stdio.h>
#include<conio.h>
void swap(int x, int y); // fn declaration
void main()
{ int a=100, b=200;
  clrscr();
  swap(a,b);    // fn call
  printf("\n Value of a: %d", a);
  printf("\n Value of b: %d", b);
  getch();
}
void swap(int x, int y) // fn definition
{
  int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Output: Value of a: 100
Value of b: 200

ii) Call by reference:

```
#include <stdio.h>
#include <conio.h>
void swap(int *x, int *y); // fn declaration
void main()
{
    int a=100, b=200;
    clrscr();
    swap(&a, &b);
    printf("\n Value of a: %d", a);
    printf("\n Value of b: %d", b);
    getch();
}
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

Output:

Value of a: 200

Value of b: 100

	Call by Value	Call by reference
i)	copies original value to function as arguments	copies address of arguments into function
ii)	changes made to parameter inside function has no effect on argument	changes made to parameter affects argument
iii)	actual & formal argument created in different memory location	actual and formal arguments created in same memory location

Passing arrays in function:

```
#include <stdio.h>
void display(int age)
{
    printf("%d", age);
}
int main()
{
    int ageArray[] = {2, 3, 4};
    display(ageArray[2]); // passing array element
    return 0;
}
```

output:

4

Advantage of user-defined function:

* program will be easy to understand, maintain & debug
* reusable codes used in other programs
* large program divided into smaller modules