Procedural Programming – I

UNIT – III

Function

Sometimes it is required to execute a group of statements in a program multiple times at different situations. For example we need to perform the addition of two numbers multiple times in the program for this we have to write the same code for number of times in the program.
This increases the program length and reduces the program efficiency.
In overcome this function concept is used.
"A function is set of instruction used to do a specific task". Every C program is collection of such function, one feature of C is that "C has collection of function".
Functions are used to divide a large program into smaller pieces.
Function can be called multiple times to provide reusability and modularity to the C program.
Block of code that performs particular task. (eg, remote)

Their two types of function-

1) library function(built in functions)
2) User defined function

Library function-

Library functions are the commonly required functions grouped together and stored in librsry. They are predefined functions and can be called anywhere e.g. printf(), scanf(), sqrt(),strlen(),getch(),clrscr() etc.

User defined function-

User defined function are the function defined by the user to do a specific task . e.g addnum(), show(), printline(), etc.

Syntax-

Basic syntax of C-function is like-

return_type function_name (data-types parameter 1, data-types parameter 2…)
{
//code to be executed
}

Example-

```
#include<stdio.h>
#include<conio.h>
void add(int , int );
void main()
{
int a,b;
printf("enter two numbers:\n");
scanf("%d \n%d",&a,&b);
add(a,b);      // function call (parameters a and b)
getch();
}
void add(int x, int y)
int c;
c=a+b;
printf("addition is:%d",c);
}
```

Output-
Enter two numbers
10
20
Addition is 30
In above program add() is a user defined function.

Advantages of user defined functions-
We can avoid rewriting same logic through functions.
We can divide work among programmers using functions
We can easily debug, test and maintain a program using functions.

Declaration-
A function is declared to tell a compiler about its existence. It is defined to get some task done.
A function is called in order to be used.
Writing a function in the program includes 3 steps-
1) function prototype
2) function call
3) function definition

Example-

```
include<stdio.h>
include<conio.h>
void hello();          //function prototype
void main()
{
hello()                // function call
getch();
}
void hello()           //function definition
{
printf("Hello world");
}
```

Output-
Hello world

1) Function Prototype
It is the declaration statement of the function. It tells compiler what type of function it is i,e return type, parameter type, etc.
It should be given above the main () function and it should have a semicolon.
The general form of the prototype statement is-

return type function name(parameter type or arguments)
{
Group statement;
Return(expression); //if returning
}

Example-
1) **Void printline()**
Void states that function is not returning any value.
Or
2) **int add(int, int)**
Int states that function is taking two values and it is returning int value.
Or

3) **char find()**
Char states that find() function is returning character value.

2) Function Call-
It is a statement using which the functions gets executed. When a function call gets encountered control jumps from main program to the function definition block. The function statements its returns to the main program and execute next to the function call. We can call function anywhere in any function or program.
A function can also be called into itself, this is known as 'recursion'.

3) Function Definition-
In the function definition we have to write the code for that function.
The function definition must be outside any function or it can be the other program also.
We can call the same function fot number of times in the program. We can call or define more than one function in a single program in a random sequence.
The function definition statement should match with the function prototype statement.

Categories of Function-
A function depending in whether arguments pass or not and whether a value is returned or not, may belong to one of the following categories.

1) function with no arguments and no return value-[void function]
When a function has no arguments and it's doesn't return anything is called as **void function**. It does not receive any data from the calling function, similarly when it does not return a value. In fact there is no transfer between the calling and called function. The complete procedure is performed by the function itself. No data communication between function.

Example-
```
include<stdio.h>
include<conio.h>
void add();                    //function prototype
void main()
{
int a,b;
add();                         // function call
getch();
}
void add()                     //function definition
int a,b,c;
printf("enter two numbers:\n");
scanf("%d \n%d",&a,&b);
c=a+b;
printf("addition is:%d",c);
}
```
2) function with arguments but no returning values-
We could make the calling function to read data from the user and pass it on the called function. These approach seems to be because the calling function can check for the validity of data if necessary before it is handed over to the called function.

Example-
```
include<stdio.h>
include<conio.h>
void add(int, int);                    //function prototype
void main()
{
int a,b;
```

```
int a,b;
printf("enter two numbers:\n");
scanf("%d \n%d",&a,&b);
add();                          // function call
getch();
}
void add(int x, int y)
{
int c;                          //function definition
c=a+b;
printf("addition is:%d",c);
}
```

Here **a, b** are the actual arguments and **x, y** are the formal arguments and. Two numbers are accepted into and b, they will be passed to the function when it is called. The value of **a** will be passed into **x** and value of **b** will be passed into **y**, similarly we can pass float or character values to the function. The name of the actual arguments and formal arguments can be same as they are considered as local variables.
The result will be printed if a=10 and b= 20 as
Addition of 10 and 20 is 30

3) function without arguments but return value.
There is no any arguments or parameters in input but returns the values.

Example-

```
include<stdio.h>
include<conio.h>
int sum()
void main()
sum()
printf("sum is: %d", c)
getch();
}
int sum()
int a, b, c;
a=10; b=20;
c=a+b;
return 0;              //OR we can also write return (c) instead of return 0;
}
```

4) with arguments and with return value-
There are arguments and also return value that's why we called that function with arguments and with return value.

Example-

```
include<stdio.h>
include<conio.h>
int sum(int, int)
void main()
{
int a,b, result;
printf("Enter a two numbers for a and b:\n");
scanf("%d \n%d",&a,&b)
result=sum(a,b);
```

```
printf("sum is :%d\n", result);
getch();
}
int sum(int x, int y)
{
int c;
c=x+y;
return c;              //return (a+b)
}
```

Output-

Enter a two number for a and b:
5
4
Sum is: 9

Variables

Local variable-

Variables which are accessed inside a function or a block are called local variables.
They can only be accessed by the function they are declared in!
Scope- scope is a region of the program where a defined variable can exist and beyond which it cannot be accessed.
The scope of this variables is only for that particular function.
Lifetime-lifetime of local variable is only till the function will be executed.

Example-

```
include<stdio.h>
include<conio.h>
int sq(int);
int y;                          //global variable (lifetime of this throughout the program end)
void main()
{
int a=5;                        //local variable (lifetime of this throughout the function end)
sq(a);
printf("square = %d",sq(a));
getch();
}
int sq (int x)
{
y=x*x;
Return y;
}
```

Global variable-

These are the variable defined outside the main method.
Global variable are accessible throughout the entire program from any function.
If a local and global variable has the same name, the local variable will take preference.
Scope- The scope of this variable is for entire body of the program.
Lifetime- lifetime of global variable is throughout the entire program till the program will be executed.

Actual and formal parameters-

When a function is called, the values that are passed in the call the arguments or actual parameters.
Formal parameters are local variables which are assigned value from the arguments when the function is called.

There are two types of method for parameter passing-
1) Call by value-
2) Call by reference

Call by value-
In this method we have to pass the value to the function by directly specifying the variable name with the function call. In this method a copy of the original arguments get created in the function and value in the actual parameter.
The example based on this method are discussed here.

Example-

```
include<stdio.h>
include<conio.h>
void add(int, int);
void main ()
{
int a,b;
printf("enter two number:");
scanf("%d %d", &a,&b);
sdd(a,b);
getch();
}
void add(int x, int y)
{
int c;
c=x+y;
printf("addition of % and %d=%d",x,y,c)
}
```

Here **a, b** are the actual arguments and **x, y** are the formal arguments. Two numbers are accepted into **a** and **b** and they will be passed to the function when it is called. The value of **a** will be value of **b** will be passed to **y**. similarly we can pass float or character values to the function. The name of the actual arguments and formal arguments can be same as they are considered as local variables.
The result will be printed if a= 10 and b= 20 as-
Addition of 10 and 20 is =30

Call by reference-
In this method the value should be passed by passing the address (reference) of variable.
In this method the original value will gets changed by the function. The variable inside the function should be of pointer type.

Example-

```
include<stdio.h>
include<conio.h>
void add(int * a, int *b);
void main()
{
int x,y,result;
 x=5;
 y=10;
result=add(&x , &y);
printf("addition is %d",result)
getch();
}
```

```
void add(int * a, int *b)
{
int c;
a=10;
b=20;
c=a+b;
return(c);
}
```

The output of above program is-
Addition is:30

The add() function interchange the values of **x** and **y** using their addresses stored in **a** and **b**. **&a** and **&b** will copy the address of **x** and **y** into the pointer a and b in the called function.

Array
All the elements of array of same data-types and hence it is called a collection of similar data elements.
Array is a collection of similar elements stored in adjacent memory location. An ordinary variable is capable of storing only one value at a time. However there are situations in which we would to store more than value at a time in single variable.
For example – suppose we have to store name or percentage of 10 students. This can be achieved by two ways-
a) by declaring a 10 variables for name and percentage.
b) by declaring such a variables which can hold the ten values.
The second way is better. It is easy to handle one variable than handling 10 variables. An array allows us to do this.

Declaration of array-
Syntax-
Data type Array_name[size/Dimension]

Example-
```
int n[5];
int salary[10];
```

here **n** is Array_name it can hold five integer values at a time. Dimension specifies the width of the array i.e. how many numbers an array can store, it should be enclosed in square brackets[]. The individual values are called elements. element number is a positive integer value and it stars with zero e.g. n[0]. n[2] will refer third element of the array, n[i] where I can take value 1,2,3,4,5, here **n** is subscripted variable (array) and **i** is its subscript.

There are two types of array-
1) One dimensional.
2) Two or multi- dimensional.

For one dimensional array we have to specify only one dimensional array. For two or multi-dimensional array we have to specify more than one dimensions. Two dimensional array is also known as matrix.

One dimensional array-
A list of items can be given none variable using only one subscript and such a variable is called a single subscripted variable or none dimensional array.
For example- int a[5];
char str[10];

in above example **a** and **str** are one dimensional array. A is integer array with five elements. **str** also one dimensional character array which can hold 10 characters i.e. one word.
When an is declared it gets defined in a consecutive (contiguous) memory locations and each location is considered as an elements of the array.

If we have array like
int num[6];
Then the computer reserves five contiguous storage location i.e.

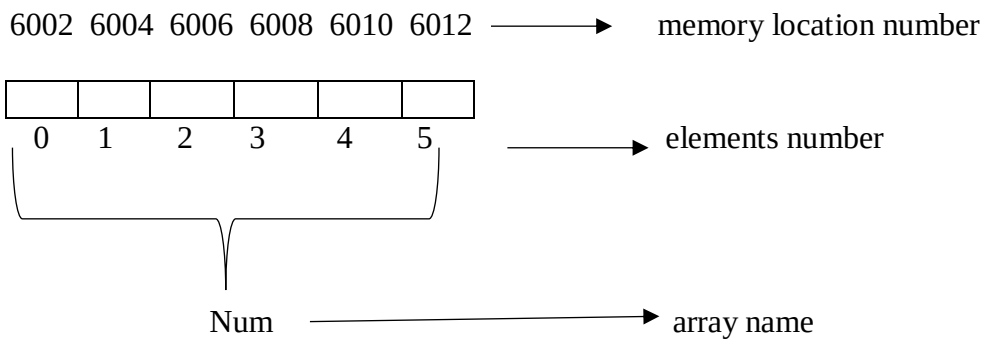

Fig- storage location in array
Memory location is given as 6002, 6004….because integer value take two bytes.

Initialization of array-
An array can be initialized in different ways. There are mainly two methods to initialize an array, initialization at the time of declaration as follows.

int num[5] = {5,10,15,20,25}
When we initialize an array at the time of declaration, it is nit necessory to specify the size of array. For example-
int num[] = {5,10,15,20,25} //is valid

In this case the size of array will be calculated by the compiler automatically.
Now the array will look like:

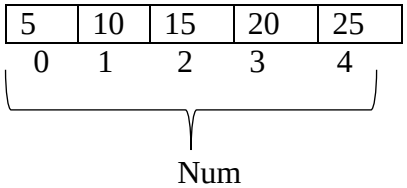

Fig- initialization of array

Using keyboard(take elements from user)-
We can initialize the array through the keyboard using scanf and for loop as follows-

```
int num[10]
printf("enter the elements in an array:");
for(i=0;i<10;i++)
{
scanf("%d",a[i]);
}
```

Similarly to scanf we can use printf and for loop to printelements of an array as follows-

```
for(i=0;i<10;i++)
{
printf("%d", a[i]);
}
```

Here every time it will accept new number and print it. Character array are manipulated in the same way as the integer array.

Two dimensinal array-
The array can have more than one dimensions also. In mathematics, we represent a particular value in a matrix by using two subscripts such as **Vij.** Here **V** denotes the entire matrix and **Vij** refers to the value in the **ith** row and **jth** cloumn. C allows us to define such table of items by using two dimensional array. A two dimensional array requires two dimensions. It is also called as matrix.

Syntax-
Data type Array_name[row-size] [column-size]

Initialization of two dimensional array-
Example-
Int num[3][3];
Char name[5][10]; //

Here num and name are two dimensional array. Num is a two dimensional array having 3*3 = 9 elements. it is also known as 3 by 3 matrix with 3 rows and 3 columns. Name is a two dimensonal character array in which first dimension is the number of elements and second dimension is the width of each element i.e. name array can hold 10 characters length.

Some more examples of array as follows-
int v[4][3] //creates a 4 by 3 matrx
int n[2][2] // creates a 2 by 2 matrix

Representation of two dimensional array-

n[2][2]	column 0	column 1
Row 0	[0][0]	[0][1]
Column 1	[1][0]	[1][1]

Program to display the number stored in two dimensional array in a matrix form.

```
include<stdio.h>
include<conio.h>
void main()
{
int n[3][3];
int i, j;
printf("\n enter the number for 3 by 3 array:");
for(i=0;i<5;i++)
{
  for(j=0;j<5;j++)
  {
    scanf("%d",[i][j]);
   }
}
printf("\n the numbers are :\n");
for(i=0;i<5;i++)
{
  for(j=0;j<5;j++)
  {
    printf("%3d", n[i][j]);
```

```
  }
 printf("\n");
}
getch();
}
```

If the number entered as 1 2 3 4 5 6 7 8 9 then they are printed in the following format:
1 2 3
4 5 6
7 8 9

The program contains two dimensional array of **3** rows and **3** columns. A nested for loop is used to accept and display numbers in each element. To vary elements **i** and **j** counter variable are used **%3d** is a formatted operator which will set the width as 3 for each element. Character \n is given outside the inner loop as the line changes for each row only, individual element can be referred as **n[i][j].**
This mechanism can be used for matrix operation like matrix addition, matrix multiplication etc.