Unit-2

Arrays, Strings and Functions.

* Arrays *

An array is a collection of variables of the same type are reffered through a common name. A specific element in an array is accessed by an index also known as subscript. In C, all arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

An array can have from one to several dimensions. Array is a collection of same type of data like int, float, char etc. Character array is known as a string.

⇒ Some properties of an Array -

1) Array is a ~~Coll~~ collection of similar datatype which stores data in continuous memory allocation. in the form of single row or column.
2) Array are also known as sub-script variable.
3) Indexing of an array always ~~Star~~ start from zero to size - 1.
4) The name of array is return the base address of an array. [Base Address - It means address of 0^{th} ($zero^{th}$) position].
5) Before use of array, array type and size must be declared.
6) The size of memory allocation of an array is calculated as follows -

   Size of memory allocation =

   size of datatype * size of array.

Prepared By:-
Chandrashekhar Verma
An IT Instructor

⇒ Types of arrays -

These are the types of arrays -

1.) **One Dimensional Array** :- The general form for declaring a single-dimension array is :-

Syntax :- (Array Declaration Syntax)

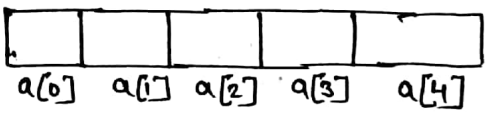
```
<datatype> <arrayName> [size] ;
```

↳ List of item can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or a one dimension array.

Like other variables, array must be explicitly declared so that the compiler can allocate space for them in memory. Here, ~~type~~ datatype declares the base type of the array, which is the type of each elements in the array, and size defines how many elements the array will hold.

Example :-

```
int age[5];
```



• Initialization an Array - Once the array has been declared, some values have to be stored ~~in~~ in it. Storing initial values in an array is called initializing an array. It is also called assiring values to an array.

It is not always necessary to specify the size of an array if it is being initialized.

Syntax -

```
<datatype> <arrayName> [size] = {value1, value2, ..... };
```

Example -

```
int a[5] = { 72, 32, 2, 4, 9 } ;
```

Note - • Till the array elements are not given any specific values they are supposed to ~~co~~ contain ~~gar~~ garbage values.

• IF the array is initialised where it is declared, mentioning the dimension of the array is optional.

• Accessing Array Elements :- A particular element in array can be accessed by specifying the array name, followed by square braces [ ], enclosing an integer, which is called the array index. The array index indicates the particular element of the array which we want to access.

Syntax -

```
<arrayName> [indexValue];
```

Example -

```
a[0] ;    // 0 is accessed.
a[1] ;    // 1 is accessed.
```

• Example :-

```
void main( )
{
  int a[5], i;
  clrscr( );
  printf("enter array elements=");
  for (i=0; i<4; i++)
      {
        scanf("%d", &a[i]);
      }
 printf("Array elements are=");
  for (i=0; i<4; i++)
  {
    printf("%d\n", a[i]);
  }
  getch( );
}
```

```
void main( )
{
  int a[5] = {10,20,30,40,50};
  /* a[0] = 10
     a[1] = 20
     a[2] = 30
     a[3] = 40
     a[4] = 50 */
  int i;
  for (i=0; i<4; i++)
  {
    printf("%d", a[i]);
  }
  getch( );
}
```

2.> Two Dimensional Array :- C supports multi-dimen-sional arrays. The simplest form of the multi-dimension array is two-dimensional array.

The declaration of a two dimensional array is similar to declaration of one dimensional array. There will be an extra set braces [ ] to indicate the 2nd index. The two dimensional arrays are declared as follow -

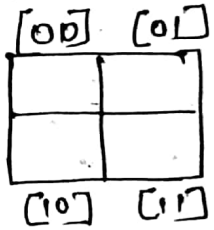
① Syntax - (2D Array Declaration Syntax)

<datatype> <arrayName> [rowsize][columnSize];

③ Two dimensional array is used to store data items in the form of row and column.

② Example -

int ary [2][2];



•> Initialization of 2D Array -

Syntax -

<datatype> <arrayName> [rowSize][columnSize] =
{(row1, column1), (row2, column2) ----- };

~~Example~~ - OR

int ary [2][2] = { {0,0}, {0,1}, {1,0}, {1,1} }

Example -

int ary [2][2] = {1, 2, 3, 4};

•> Accessing 2D Array Element -

Syntax -

<arrayName> [rowIndex] [columnIndex];

Example -

```
int ary [2][2];
ary[0][0] = 1;
ary[0][1] = 2;
```

Prepared By:-
Chandrashekhar Verma
An IT Instructor

•> Example :-

```
void main()
 {int a[2],[2], i, j;
  clrscr();
  printf("enter array elements=");
  for(i=0; i<2 ; i++)
    {
      for(j=0 ; j<2; j++)
        {
          scanf("%d", &a[i][j]);
        }
    }
  printf(" Array elements :- ");
  for(i=0; i<2; i++)
  {
  for(j=0; j<2 ; j++)
    {
      printf("%d\n", a[i][j]);
    }
  }
  getch();
}
```


3.> Multi-Dimensional Arrays - It is possible to use array of three or more dimensions. The exact limit ~~of~~ ~~is~~ is determined by the compiler. A three dimensional array can be thought of as an ~~of~~ arrays of arrays.

Syntax :-

⟨datatype⟩ ⟨arrayName⟩ [size1] [size2] ---- [size-m];

=> Advantages of Using Array -

(1.) One advantages of using arrays is that, all elements are referred to collectively by a single-name, the array name. Otherwise we would have different variable names for each individual data.

(2) We can also define two or more dimensional arrays, provided the programming language that we use allows its usage.

(3) Once declaration and initialization is done, we can sort the data and also search for data in the array.

=> Limitations of Array -

(i) unused memory. (ii) insufficient memory

(iii) To store or to access the data items, one loop is required. This is because one-dimensional array has different memory spaces or blocks sequentially. Hence, it is needed to reach at every position one by one.

* Strings *

A group of integers can be stored in an integer array, similarly a group of characters can be stored in character array. many time character arrays are also called strings.

Character arrays or strings are used by programming languages to manipulates text such as words and sentences. A strings constant is a one-dimensional array of characters terminated by a null ('\0').

Each character in the array occupies one byte of memory and the last character is always '\0'. The terminating null ('\0') is important because it is the only way the functions that work with a string can know where the string ends. A string not terminated by a '\0' is not really string but merely a collection of characters.

=> String Declaration -

Syntax - i) char <arrayName> [size];

ii) char <arrayName> [ ];

Example :- i) char str[20];

iii) char str[ ];

Difference in above declaration-

(i) When we declare char as "str[20]", 20 bytes of memory space is allocated for holding the string value.

(ii) When we declare char as "str[ ]", memory will be allocated as per the requirement during execution of the program.

⇒ String Initialization -

Whenever we declare a string then it will contain garbage values inside it. we have to initilize string or character array before using it. Process of assigning some legal default data to string is called Initialization of String. There are different ways to initializing string in C programming -

(1) Initializing Unsized Array of Character-

Unsized array, in this array length (size) is not specified while initializing character array using this approch. Array length is automatically calculated by compiler.

Individual character are written inside single quotes, separated by comma to form a list of character. Complete list is wrapped inside pair of curly braces.

NULL character should be written in the list at the last because it it ending or terminating character in the string / character array.

Example -

```
char name[ ] = { 's','a','i','\0' };
```

(2) Directly • initialize String variable -

In this method we are directly assigning string to variable by writing text in double quotes.

In this type of initialization, we won't need to put NULL or Ending / terminating character at the end of string. It is appended automatically by the compiler.

Example -

```
char name[ ] = "sai";
```

Note :- The sized array can be declared as follows-

(a) `char name[10] = {'s', 'a', 'i', '\0'};`

(b) `char name[10] = "sai";`

Both declaration is fixed length size. We can only put to character ~~array~~ string in this type of character array.

(3) Character Pointer Variable -

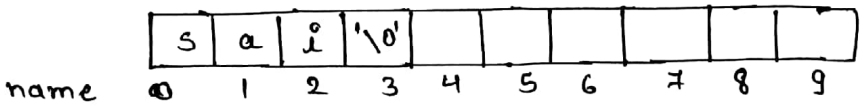
Declare character variable of pointer type so that it can hold the base address of "string". Base address means address of first array element i.e. address of name[0]. Null character is appended automatically.

Example -

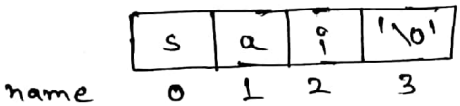
```
char *name = "sai";
```


⇒ Memory Representation -

(a) `char name[10] = "sai";`



(b) `char name[ ] = "sai";`



⇒ String Handling Library Functions :-

There are many useful strings handling library functions provided by every C compiler which can be used to carry out many of the string manipulations. If we want to use string functions in C we need to include string.h header file in our program. The String Library functions are -

(a) gets() - This function is used to take input a one line string from the keyboard. gets() function is available in stdio.h header file.

Syntax :- `gets(stringVariableName);`

(b) puts() :- This function is used to print string on the screen. It is available in stdio.h header file.

Syntax :-

```
puts(string variableName);
```

Example –

```
#include <stdio.h>
#include <conio.h>
void main()
{
   char n[50];
   clrscr();
   printf("enter a string : ");
   gets(n);
   puts(n);
   getch();
}
```


(c) strlen() :- We can find the length of any string by using strlen() function. means this function counts the number of character present in a string.

Syntax –

```
integerVariable = strlen(string);
```

Example –

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
  char x[] = "Hello";
  int s1, s2;
  clrscr();
  s1 = strlen(x);
  s2 = strlen("Hello World");
  printf("String = %s, length = %d", x, s1);
  printf("String = %s, length = %d", "Hello World", s2);
  getch();
}
```

O/P –
String = Hello, length = 5
String = Hello World, length = 11

(d) strcpy() :- This function copies the contents of one string into another. The base addresses of the source and target strings should be supplied to this function. This function works almost like string assignment operator

Syntax :-

```
strcpy(string1, string2);
```
(copy)

(64.) In this function the string1 is a 1st argument it is a string ~~variable~~ is variable in which the content of string2 is copied. It means it is destination The another argument string2 is a string which content is copied into string1.

Example :-

```
void main()
{
 char str1[ ]="Hello", str2[ ]="World";
 clrscr();
 strcpy (str1, str2);
 puts (str1);
 getch();
}
```

O/p :- World

(e) strrev() :- This function is used to reverse the argument string. It take one argument as string & reverse it.

Syntax :-

```
strrev (StringVariableName);    or
strrev ("string");
```

(f) strlwr() :- It is used to convert all alphabets in a string to lower case letters.

Syntax -

```
strlwr (StringVariableName);
        or
strlwr ("string");
```


(g) strupr() :- It is used to convert all alphabets in a string to upper case letters.

Syntax :- strupr (StringVariableName); OR
strupr ("string");

Example -

```
void main()
{
charstr1[ ]="Hello";
 puts(strrev(str1));
 puts (strrev ("world"));
 puts (strlwr (str1));
 puts (strlwr ("World"));
 puts (strupr (str1));
 puts (strupr ("World"));
 getch();
}
```

O/p :- olleH
dlroW
hello
world
HELLO
WORLD

(h) strcat() :- This function concatenates the source string at the end of the target string.

Syntax - strcat(string1, string2).

String2 added at the end of string1. ~~and removed~~ without any blank space.

(i) strcmp() :- This is a function which compares two strings to find out whether they are same or different. The two string are compared character by character until there is a mismatch or end of one of the string is reached, whichever occurs first.

If the two strings are identical, it returns a value zero ~~value~~. If they are not identical, it returns the numeric difference between the ASCII values of the first non matching pairs of character.

Syntax -

IntegerVariable = strcmp (string1, string2)

Example:-

```
void main()
{
  Char  str1[] = "hello", str2[] = "world" ;
  int i,j,k;
  clrscr();
  i= strcmp (str1,str2);
  j= strcmp (str1, "hello");
  k= strcmp (str2, str1);
  printf("%d %d %d \n", i, j, k);
  puts(strcat (str1, str2));
  getch();
}
```

O/p :-

-15 0 15

Helloworld

(j) strchr() :- This function is used to find a character in a given string. This function contains two argument 1st is a string, & 2nd is a character which is need to find (search) in string. If character is find it returns zero otherwise it returns non zero value.

Syntax -

strchr (string, char);

(69.) (k) strstr() :- strstr() is used to find a sub string from given string. It return zero if sub string is found otherwise it return non-zero.

Syntax - strstr(string1, string2);

(l) strcmpi() :- This function is used to compare two string without checking its case. (i denote that this function ignores case)

Syntax - IntegerVariable = strcmpi(string1, string2);

Example - m = strcmpi("DELHI", "Delhi");

Output - m=0.

(m) strncmp() :- To compare the first n characters of two string.

Syntax - IntegerValue = strncmp(string1, string2, n);

Example - m = strncmp("DELHI", "DIDAR", 2);

Output - m = -4.

(n) strncat() :- To join specific number of letters to another string. Appends first n characters of a String at the end of another.

Syntax - strncat(string1, string2, n);

Example -
```
char s1[10] = "New";
char s2[10] = "Delhi-41";
strncat(s1,s2,3);
```

Output - s1 will be "NewDel".

* Functions *

* What is function in C?

A function is a self-contained, group of statements that take inputs and together preform some specific task & Every C program has at least one function which is main().

C functions are basic building blocks in a program. All C programs are written using functions to improve re-usability, understandability and keep track on them.

A large problem has to be split into smaller segments so that it can be efficiently solved. A function in a set of statements that takes inputs, do some specific computation and produces output.

The Function in C language is also known as procedure or subroutine in other programming langu. To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

* Uses / Advantages of functions.

1.) Modular and Structural programming can be done.
2.) It follows Top-Down Execution approch, so main can be kept very small.
3.) Individual functions can be easily built & tested.
4.) Program development become easy.
5.) Frequently used functions can be put together in the customized library.
6.) A function can call other function & also itself.
7.) It is easier to understand the program topic.
8.) Functions are very useful when a block of statement has to be written / executed again & again.
9.) Functions are useful when program size is too large or complex.
10.) It is also used to reduce difficulties during debugging a program.

* Types of C function

There are two types of function in C programming -

(i) Library or BuiltIn or Pre Defined or System Defined.

(ii) User Defined functions.

(4.)

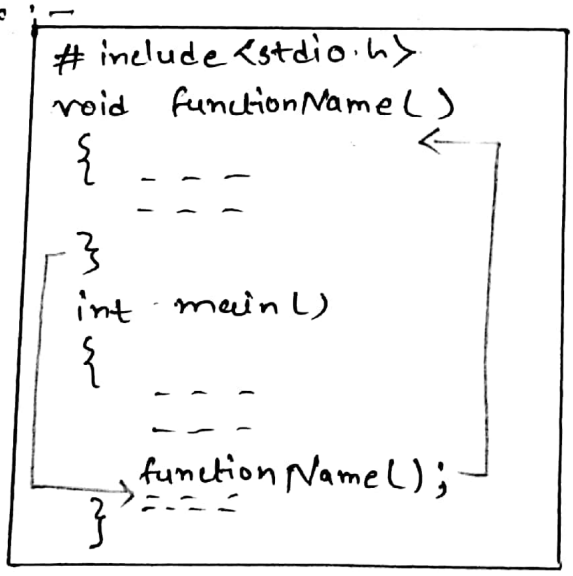
1.) Standard Library Functions - These are built in functions in C to handle tasks such as mathematical computations, I/O processing, string handling etc.

These functions are available along with the compiler and are used along with the required header files. Example - printf(), scanf() etc.

2.) User Defined Functions - C language allows programmer to define function. Such functions are called user defined functions. Depending upon the complexity & requirement of the program you can create as many user defined functions as you want.

User Defined functions are self contained blocks of statements which are written by the user to compute a value or to perform a task. They can be called by the main program repeatedly as per requirement.

Syntax :-



The execution of a C program ~~begins~~ begins from main() function. When a compiler encounters functionName(); inside the main function, control of the program jumps to void functionName(), and the compiler starts executing the codes inside the user defined function. The control of the program jumps to statement next to functionName(); once all the codes inside the function are executed

The advantages of user defined functions are -

(i) The program will be easier to understand, maintain & debug.

(ii) Reusable codes that can be used in other programs.

(iii) A large program can be devided into smaller modules. Hence a large project can devided among many programmers.

* The C Function Concepts.

(1) Function Declaration or prototype:- This informs compiler about the function name, function parameters and return type.

A function declaration tells the compiler about function name and how to call the function.

Syntax -

```
return_type function_name (parameter list);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

(2) Function Defination :- A function defination provides the actual defination or body of the function to the compiler.

Syntax -

```
return_type functionName (parameter list)
{
    body of function;
}
```


All the parts of functions are as follows -

(i) Return Type - A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case the return_type is the keyword void.

(ii) Function Name – This is the actual name of the function.

(iii) Parameters – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is reffered to as actual parameter or ~~eg~~ argument. The parameter list refers to the type, order & number of parameters of a function. Parameters are optional that is a function may contain no parameters or ~~eg~~ arguments.

(iv) Function Body – It contains a collection of statements that define what the function does.

(3) Function Calling :– To use a function, we call that function to perform the defined task. When a program call a function, the program control is transferred to the called function. A called function preforms a defined task and when its return statement is executed, it returns the program control back to the main program.

Syntax –

```
functionName (argument list);
```

To call a function, pass the actual parameters along with the function name, and if the function returns a value, then you can store the returned value.

Example –

```
#include<stdio.h>
float square(float x); // function declaration
int main()
{
    float m, n;
    printf("enter two numbers : ");
    scanf("%f", &m);
    n = square(m); //function call
    printf(" square = %f", n);
}
float square (float x) // function defination
{
    float p = x*x;
    return (p);
}
```

* Rules for Writing Function in C programming

1.) C program is a collection of one or more functions A function gets called when the function name is followed by a semicolon.

2.) A function is defined when function ~~main~~ name is followed by a pair of braces in which one or more statements may be present.

3.) Any function can be called from any other function main also.

4.) A function can be called any number of times.

5.) The order in which the functions are defined in a program and the order in which they get called need ~~to~~ not necessarily be same.

6.) A function can call itself (process of recursion).

7.) A function can be called from other function but a function cannot be defined in another function.

* How to Call functions / Function Arguments / Passing Argument to Function.

If a function is use arguments, it must declare variable that accept the values of the arguments. These variables are called the formal parameters of the function.

Formal parameters behave like other local variable inside the function and are created upon entry into the function and destroyed upon exit.

The two ways in which arguments can be passed to a function are :-

(i) Call by value. (ii) Call by reference.

1.) Call by value - In this method the value of the variable is passed to the function as parameter. The value of the actual parameter can't be modified by formal parameter.

Different memory is allocated for both actual and formal parameters because value of actual parameter is copied to formal parameter.

By default C uses call by value to pass arguments. The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameters of the function. In this case changes made to the parameters inside the function have no effect on the argument.

Example :-


```
#include <conio.h>
#include<stdio.h>
void swap (int x, int y);
void main()
{
   int a=10, b=20; clrscr();
   printf(" Before swap : a= %d \t b=%d \n", a, b);
   swap (a, b);
   printf(" After swap : a=%d \t b=%d\n", a, b);
   getch();
}
void swap (int x, int y)
{
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```

O/p :-

after swap a=10 b=20

before swap a=10 b=20.

2.) Call by reference - In call by reference, the address of actual argument is passed to formal argument of the called function.

This means by accessing the addresses of actual arguments we can alter them without from the called function. This means that changes made to the parameter affect passing the argument. To pass the value by reference, argument reference is passed to the functions just like any one value. The value of actual parameter can be modified by formal parameter, same memory is used for both actual & formal parameters since only address is used by both parameters.

Example

```
#include <stdio.h>
#include <conio.h>
void swap (int *x, int *y);
void main()
{
  int a=10, b=20;
  clrscr();
  printf("Before Swap a=%d \t b=%d", a, b);
  swap (&a, &b);
  printf("After swap a=%d \t b=%d", a, b);
  getch();
}
void swap (int *x, int *y)
{
  int temp;
  temp = *x;
  *x = *b;
  *b = temp
}
```

O/p:-

Before swap a=10. b=20
After swap a=20 b=10.

* Actual and Formal Parameters in function.

Actual Parameters — This is the argument which is used in function call. (or which is used to pass in function)

Formal Parameters — This is the argument which is used in function defination.

* The main() Function.

When the operating system runs a program in C it passes control of the computer over to that program. In C language program, it's the main() function that the operating system is looking for. The main function looks like -

```
main ()
{
  - - -
  - - -
}
```


The main() function uses its parantheses to contain any information typed after the program name at command prompt. They contain programming

instruction that belong to the function. The main() function has several ~~specific~~ special ~~to~~ properties are as follows -

(i) In particular, it cannot be called recursively.

(ii) Its address cannot be taken.

(iii) It cannot be predefined and can not be overloaded.

(iv) It can not be ~~defined~~, delated or declared.

* Difference between Call by Value and Call by Reference

S.No.	Call By Value	Call By Reference.
1.	This is a usual method to call a function in which only the value of variable is passed as an argument.	In this method, the address of the variable is passed as an argument.
2.	Any alteration in the value of the argument passed is local to the function and is not accepted in the calling program.	Any alteration in the value of the argument passed is accepted in the calling program (since alteration is made indirectly in the memory location using the pointer).
3.	memory location occupied by formal & actual argument is different.	memory location occupied by formal & actual arguments is same & there is a saving of memory location.
4.	Since a new location is created, this method is slow.	This method is fast.
5.	There is no possibility of wrong data manipulation since the arguments are directly used in an application.	There is a possibility of wrong data manipulation since the addresses are used in an expression. A good skill of programming is required here.
6.	At the time of function calling a programmer can send a copy of variable of value.	User can send address of variable.
7.	All the operation are performed on values & no use of pointer operators.	All operation are performed on address & pointer operators (&, *) are used.
8.	~~memory location used by formal & actual arguments are different.~~	~~memory location used by formal & actual arguments are same~~
9.	Function Declaration :- int sum (int, int);	Function declaration - int sum (*int*, *int*);

* Types of Function Arrangement / Categories of function.

All C Functions can be called either with arguments or without arguments in a C program. These functions may or may not return values to the calling function. C supports the following four categories of it -

(i) Function with arguments and with return value.
(ii) Function with arguments and without return value.
(iii) Function without arguments and without return value
(iv) Function without arguments and with return value.

1.) Function without argument and without return value -

When a Function has no arguments, it does not receive any data from the calling function. Similarly when it does not return a value, the calling function does not receive any data from the called function. There is no data transfer between the calling function and the called function.

- function declaration :-

```
void functionName();
```

- function call :-

```
functionName();
```

- function defination :-

```
void functionName()
{
    statements;
}
```


Example :-

```
#include <stdio.h>
#include <conio.h>
void sum();  // Function Declaration
void main()
{
   clrscr();
   sum();   // Function call.
   getch();
}
```

```
void sum ()      // Function Defination
{
  int x, y, sum=0;
  printf(" enter two values = ");
  scanf("%d%d", &x, &y);
  sum= x+y;
  printf(" Sum = %d", sum);
}
```

2.) Function with Argument and Without Return Value -

In this case, a function has argument, it receives data from the calling function. Function processes the data but result or any other value is not returned to the calling function. It is single (one-way) Type Communication. Generally output is printed in the called function.

- Function declaration :-
  void function (datatypes);
- function Call :-
  function (variables);
- function Defination :-
  void function (datatype1 variable1 ...)
  {
  statements;
  }

Example -

```
#include<stdio.h>
#include<conio.h>
void sum (int, int);  // Function Declaration
void main()
{ int x, y;
  clrscr();
  printf("enter two numbers =");
  scanf("%d%d", &x, &y);
  sum (x, y);  // Function Call with actual argument
  getch();
}
void sum (int a, int b) // Function defination with
                        formal argument
{ int sum=0;
  sum= a+b;
  printf(" sum =%d", sum);
}
```

3.) Function with Arguments and with Return value –

In this case a function has arguments, it receives data from the calling function. Function processes the data & result or any other value is returned to the calling function. It can be termed as Two-way communication between calling function & called function.

(80)

- function declaration :-

```
returntype functionName (datatypes);
```

- function call :-

```
functionName (variables);
```

- function defination :-

```
return type functionName (datatype1 variable1 ---)
{
  statements ;
  return (variable);
}
```


Example –

```
#include<stdio.h>
#include<conio.h>
int ~~void~~ sum (int, int); // Function declaration
void main()
{
  int x,y, sum1=0;
  clrscr();
  printf("enter two numbers =");
  scanf("%d%d", &x,&y);
  sum1= sum(x,y);  // Function call
  printf(" sum = %d", sum1);
  getch();
}
int sum (int a, int b) // function defination
{
  int s=0;
  s=a+b;
  return (s);
}
```

4.) Function without arguments and with Return value –

In such case we need to design function that may not take any argument but returns a value to the calling function. A difficult example is a getchar() function

which is not take any argument but it return a character value.

- function declaration :-

```
returntype functionName( );
```

- function call :-

```
functionName( );
```

- function defination :-

```
returntype functionName ( )
{
  statements ;
  return (variable);
}
```

Example -

```
#include <stdio.h>
#include <conio.h>
int sum ( ) ;   // Function declaration
void main ( )
{
   int s=0;
   clrscr ( );
   s = sum ( );
   printf (" sum = %d", s);
   getch ( );
}
int sum ( )
{
   int a, b, sum1 = 0;
   printf (" enter two values =");
   scanf ("%d%d", &a, &b);
   sum1 = a+b;
   return (sum1);
}
```


* Nesting of function

C permits Nesting of function freely main in call function1 which can call function2 which can call function3 and so on. There is in principal no limit as to how deeply function can be Nested. Calling one function from within another function is said to be nesting of function calls.

Example -

```
#include <stdio.h>
#include <conio.h>
void disp1();
void disp2();
void disp3();
void main()
{
  clrscr();
  printf("main function\n");
  disp1();
  getch();
}
void disp1()
{
  printf("Disp1 function\n");
  disp2();
}
void disp2()
{
  printf("Disp2 function\n");
  disp3();
}
void disp3()
{
  printf("Disp3 function\n");
}
```


* Recursive Function / Recursion

Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied. The process is used for repetitive computations in which each action is stated in terms of a previous result. A function calling itself again & again to compute a value is referred to recursive function or recursion. many iterative or repetitive problems can be written in this form.

To solve a problem recursively, two condition must be specified -

i) The problem must be written in a recursive form.
ii) The problem must include a stopping condition.

Recursive function can be effectively used to solve problem where the solution is expressed in terms of successively ~~applying~~ applying the same solution to subsets of the problem.

When we write recursive function, we must have an "if-statement" somewhere to force the function to return without recursive call being executed. Otherwise the function will never return.

⇒ Use of Recursive Function -

(i) Recursive functions are written with less number of statements co-passed function.

(ii) Recursion is effective where terms are generated successively to compute a value.

(iii) Recursion is useful for branching process. Recursion helps to create short code that would otherwise be impossible.

In C recursive function call can be used in two ways, function call can either be direct or indirect.

Example- Direct

```
F1 ()
{
  ---
  ---
  F1();
} ---
```

Indirect

```
F1()
{ ---
  F2();
  ---
}
F2()
{ ---
  ---
  F1();
  ---
}
```

Example :-

```
int fact (int n)
{ int f=1;
  if (n==0)
     f=1;
  else
     f= n* fact(n-1);
  return (f);
}.
```

* Functions with Arrays (passing array to function) / Function Using Arrays -

It is possible to pass the value of an array to a function. To pass an array to a called function, it is sufficient to list the name of the array, without any subscript and the size of an array as arguments. The called function expecting this call must be appropriately defined.

Example -

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

float avg (float age[]);
int main()
{
   float avt , age [ ] = { 23, 55, 22, 3, 40, 18 };
   clrscr();
   avt = avg (age);
   printf (" Average age = %.2f", avt);
   return 0;
}
float avg (float age[])
{
   int i;
   float a, sum=0.0;
   for (i=0; i<6 ; ++i)
   {
      sum = sum + age[i];
   }
   a = (sum/6);
   return (a);
}
```


Output :-

Average age = 26.84

* Passing string to function / Functions with String -

Strings are character arrays. So they can be passed to a function in similar manner as array.

⇒ Rules for passing ~~string~~ array/string to function :-

(i) The string to be created must be declared as a formal argument of function.

Example -

```
void disp (char name[ ]);
```

(ii) The function prototype must show that argument is a string for the function defination. the prototype can be written as. -

```
void disp (char name[ ])
{
  - - -
}
```

(iii) A called to the function must have a string array name without subscript as its actual argument.

```
disp (name);
```

Example :-


```
#include <stdio.h>
#include <conio.h>
void disp (char str[ ]);
void main ()
{
   char str[50];
   clrscr ();
   printf ("Enter string = ");
   gets (str);
   disp (str);
   getch ();
}
void disp (char str[ ])
{
   printf ("string is = ");
   puts (str);
}.
```

O/P:-
Enter string = Hello
String is = Hello.

* Scope and Lifetime of Variable -

•) Scope :- The scope of any variable is actualy a subset of life time. A variable may be in the memory but may not be accessible though. So, the area of our program where we can actually access out entity (variable in this case) is the scope of the variable. [Area where a variable can be accessed is known as scope of variable.]

•) Life Time :- Life time of any variable is the time for which the particular variable outlives in memory during running of the program. [The time period for which a variable exists in the memory is known as lifetime]

•) The scope of any variable can be broadly categorized into three categories :-

1) Global scope - when variable is define outside all functions. It is then available to all the functions of the program and all the blocks program contains.

⇒ Scope of Global Variable :- These variable are globally accessed from any part of the program. These are ~~declared~~ declared before main() function.

⇒ Lifetime of Global Variable :- Global variables exists in the memory as long as the program is running. These variables are destroyed from the memory when the program terminates. These variables occupy memory longer then local variables.

2) Local Scope - when varaible is defined inside a function or a block, then it is locally accessible within the block and hence it is a local variable.

⇒ Scope of Local Variable - Local variable can be used only in the function in which it is declared.

⇒ Lifetime of Local Variable - The time period for which a variable exists in the memory is known as life-time of variable. Lifetime of local variable starts when control enters the function in which it is declared and it is destroyed when control exists from the function.

3.) Function Scope - When a variable is passed as formal arguments, it is said to have function scope. ~~Formal~~ In the defination of function parameters which are called formal parameters.

Formal parameters, are treated as local variables ~~data~~ with-in a function and they take precedence over global variables.

Example -

```
# include <stdio.h>
# include <conio.h>
int global = 100;        // Global Variable Declaration
void funct();
void main()
{
   int local=10;         // Local Variable Declaration
   printf(" Global value = %d", global);
   printf(" Local Value = %d", local);  funct();
   getch();
}
void funct()
{
   printf(" Global = %d", global);
}
```

O/p :- Global Value = 100
Local value = 10
Global = 100.

* Command Line Argument in C.

If any value is passed through command prompt at the time of running of program is known as command line argument. It is mostly used when you need to control your program from outside. Command line arguments are passed to main() function. main() function of a C program accepts arguments from command line or from other shell scripts.

⇒ When use Command Line Argument — When you need to developing an application for DOS operating system then in that case command line arguments are used. DOS OS is command interface operating system so by using command we execute the program with the help of command line arguments we can create our own commands.

⇒ In Command line arguments application main() function will takes two arguments that is ;—

(i) argc :- argc is an integer type variable and it holds total number of arguments which is passed in main() function. It take number of arguments in the command line including program name.

(ii) argv[] :- argv[] is a char* type variable which holds actual arguments which is passed to main function.

⇒ Syntax of main() function when using Command Line Arguments —

```
int main(int argc, char *argv[])
{
    statements;
    - - -
    - - -
}
```

⇒ Compile and Run Program when using Command Line Arguments —

Command line arguments are not compile and run like normal C programs, these programs are Compile & run on command prompt. To compile and Link Command Line program we need TCC Command. The steps can be —

(i) First open Command prompt.

(ii) Follow you directory where your code saved.

(iii) For Compile :- C:/TC/BIN> TCC <filename>.C ↵

(iv) For Run :- C:/TC/BIN> <fileName> <arguments> ↵

(89) Example :-

Compile :-

C:/TC/BIN> TCC myemd.c ↵

Run :-

C:/TC/BIN> myemd 10 20 ↵

Here myemd.c is your program file name & TCC is a command. In "myemd 10 20" statement we pass two arguments.

⇒ Example :- [save As - myemd.c]

```
#include<stdio.h>
#include<conio.h>
void main (int argc, char * argv[])
{
  int i;
  clrscr();
  printf(" Total Number of arguments : %d", argc);
  for (i=0; i<argc ; i++)
  {
    printf(" %d argument : %s", i, argv[i]);
  }
  getch();
}
```

Output :-

```
C:/TC/ BIN> TCC myemd.c // Compile Program.
C:/TC/ BIN> myemd 10 20 // Run program with two argument
Total number of arguments :3
0 argument : C:\TC\BIN\myemd.exe  <- 1st Argument is program name
1 argument : 10                   <- Arguments
2 argument : 20                   <- Arguments
```

Note :- Here show 3 arguments because argc take number of arguments in the command line including program name.

⇒ Why command line arguments program not directly run from TC IDE :- Command line argument related programs are not execute directly from TC IDE because arguments can not be passed.

- Whenever the program is compiled and link we will get .exe file & that .exe file itself is command. To load the application into the memory we required to use program name & command name.
- argc and argv are local variable to main function because those are the main() function parameters.
- Accourding to the ~~prot~~ storage class of C argc & argv are auto variable to main() function, so we can not extend the range of auto variable.
- By using argc and argv we can not access command from data outside of the main() function. We ~~are~~ required to access command from data outside of the main() function then use _argc and _argv & variable. _argc and _argv are global variable which is declared in dos.h.

* Maths and Character functions -

⇒ Character Functions :- There are many inbuilt functions in C language which ~~can~~ are used to validate the data type of given variable. List of inbuilt character validation function in C programming language are ~~is~~ supported by ~~cty~~ ctype.h header file. The functions are as follows -

(1) isalpha() :- it checks whether given character is alphabet or not. Syntax :-

```
int isalpha (char variable);
```

(2) isdigit() :- It checks whether given character is digit or not. Syntax -

```
int isdigit (char variable);
```

(3) isalnum() :- It checks whether given character is alphanumeric or not. Syntax -

```
int isalnum (char variable);
```

(4) isspace() :- It check whether given character is space or not. Syntax -

```
int isspace (char variable);
```

(5) islower() :- It checks whether given character is in lowercase or not. Syntax -

```
int islower (char variablename);
```

(6) isupper() :- It checks whether given character is in uppercase or not. Syntax -

```
int isupper (char variable);
```

(7) isxdigit() :- It checks whether given character is hexadecimal or not. Syntax -

```
int isxdigit (char variable);
```

(8) ispunct() :- It checks whether the given character is punctuation character (mark) or not. Syntax -

```
int ispunct (char variable);
```

(9) tolower() :- It 1st checks the inputed character is in lower case or not. If it is in lowercase this function do nothing, otherwise it converts the inputed character into lower case. Syntax -

```
int tolower (char variable);
```

(10) toupper() :- It 1st checks the inputed character is in upper case or not. If it is uppercase this function do nothing, otherwise it converts the inputed character into upper case. Syntax -

```
int toupper (char variable);
```

Note :- All the character functions have return type int, They returns integer value (zero or one) based on their functionality.

•> Math Functions :-

(1.) abs() function - It returns the absolute value of an integer. The absolute value of a number is always positive. Only integer values are supported in this function.

stdlib.h & math.h header files support the abs() function. Syntax -

```
int abs (int variable);
```

~~abs returns type is integer~~.

(2.) floor() function - It returns the nearest integer value which is less than or equal to the floating point argument passed to this function. Syntax -

```
double floor (float variable);
```

(3.) round() function - It returns the ~~nearest~~ nearest integer value of the float / double / long double argument passed to this function.

If decimal value is form ".1 to .5" it returns integer value less than the argument. If decimal value is from ".6 to .9", it returns the integer value grater than the argument. Syntax -

```
double round (double variable);
float roundf (float variable);
long double roundl (long double variable);
```

(4) ceil() function - It returns the nearest integer value which is graterthan or equal to the argument passed to this function. syntax -

```
double ceil (float variable);
```

(5) trunc() function - It truncates the decimal value from floating point value and returns integer value. Syntax -

```
double trunc (double variable);
float trunc (float variable);
long double truncl (long double variable);
```

(6) sin(), cos(), tan(), exp(), log() :- sin(), cos(), and tan() function in C are used to calculate sine, cosine, and tanget values.

(93)

- sinh(), cosh(), tanh() functions are used to calculate hyperbolic sine, cosine and tangent values.
- exp() function is used to calculate the exponential "e" to the xth power.
- log() function is used to calculates natural logarithm and log10() function is used to calculates base 10 logarithm.

(7) sqrt() function :- It finds the square root of the given number. syntax -

```
double sqrt (double variable);
```

(8) pow() function :- It finds the power of a given number.
syntax -

```
double pow (base, exponent);
```

→ A program to explain character functions.

```
# include <stdio.h>
# include <conio.h>
# include <ctype.h>
void main()
{
  char ch;
  clrscr();
  printf ("enter a character : ");
  scanf ("%c", &ch);
  if (islower (ch))
  {
    printf ("\n character is in lowercase");
    printf ("\n uppercase character is %c", toupper (ch));
  }
  if (isupper (ch))
  {
    printf ("\n character is in uppercase");
    printf ("\n lowercase character is %c", tolower (ch));
  }
  if (isdigit (ch))
  { printf ("\n character is a digit");
  }
```

```
if (isalnum(ch))
    printf("character is alphanumeric");
if (isalpha(ch))
    printf("character is alphabet");
if (isspace(ch))
    printf("character is space");
if (ispunct(ch))
    printf("character is punctuation mark");
if (isxdigit(ch))
    printf("character is hexadecimal");
getch();
}
```

Output :-

```
enter a character : w ↵
character is in lowercase.
Upper case character is W
character is alphanumeric
character is alphabet.
```

•> A program to explain math functions.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
  float n1;
  int n2, b, e;
  clrscr();
  printf("enter a number : ");
  scanf("%f", &n1);
  printf("\n sine value = %.2f", sin(n1));
  printf("\n cosine value = %.2f", cos(n1));
  printf("\n tangent value = %.2f", tan(n1));
  printf("\n hyperbolic sine value = %.2f", sinh(n1));
```

```
printf("hyperbolic cosine value = %.2f", cosh(n1));
Printf("\n hyperbolic tangent value = %.2f", tanh(n1));
printf("\n\n\n Enter a number : ");
scanf("%d", &n2);
printf("\n Square root of a number is : %d", sqrt(n2));
printf("\n\n\n enter base & power value : ");
scanf("%d%d", &b, &e);
printf("Answer = %d", pow(b,e));
getch();
}
```

— x —