ARRAYS&FUNCTIONS

ARRAYS

Arrays are data structures which hold multiple variables of the same data type. An array is an identifier to store a set of data with common name. Note that a variable can store only a single data. Arrays may be one dimensional or multi dimensional.

Arrays, like other variables in C, must be declared before they can be used.

```
int names[4];
names[0] = 101;
names[1] = 232;
names[2] = 231;
names[3] = 0;
```

We created an array called names, which has space for four integer variables.

Arrays have the following syntax, using square brackets to access each indexed value (called an element).

x[i]

so that x[5]refers to the sixth element in an array called x. In C, array elements start with 0. Assigning values to array elements is done by,

x[10] = g;

and assigning array elements to a variable is done by,

g = x[10];

Defining an array one dimensional arrays

Definition: Arrays are defined like the variables with an exception that each array name must be accompanied by the size (i.e. the max number of data it can store).For a one dimensional array the size is specified in a square bracket immediately after the name of the array.

The syntax is

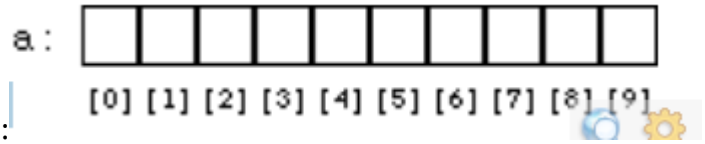
data-type array name[size];

So far, we've been declaring simple variables: the declaration **int i;** declares a single variable, named i, of type int. It is also possible to declare an arrayof several elements. The declaration

int a[10];

declares an array, named a, consisting of ten elements, each of type int. We can represent the array a

above with a picture like this:



Array Initialization

Although it is not possible to assign to all elements of an array at once using an assignment expression, it is possible to initializesome or all elements of an array when the array is defined. The syntax looks like this:

```
int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

The list of values, enclosed in braces {}, separated by commas, provides the initial values for successive elements of the array.

If the statement is like
int x[6]={0,1,2};
then the values are stored like x[0]=0, x[1]=1, x[2]=2, x[3]=0, x[4]=0 and x[5]=0.

Processing one dimensional array

1)Reading arrays: For this normally we use for- loop. If we want to read n values to an array name called'mark' , the statements look like

```
int mark[200],i,n;
for(i=1;i<=n;++i)
scanf("%d",&x[i]);
```

Note: Here the size of array declared should be more than the number of values that are intended to store.

2) Storing array in another:

To store an array to anotherarray. Suppose a and b are two arrays and we want to store that values of array a to array b. The statements look like

```
float a[100],b[100];
int I;
for(i=1;i<=100;++i)
b[i]=a[i];
```

Problem:To find the average of a set of values.

```
#include<stdio.h>
main( )
{
intx,i;
float x[100],avg=0;
printf("\n the no: of values ");
scanf("%d",&n);
printf("\n Input the numbers");
for(i=1;i<=n;++i)
{
scanf("%f",&x[i]);
avg=avg+x[i];
}
avg=avg/n;
printf("\n Average=%f",avg);
}
```

MULTI-DIMENSIONAL ARRAYS

Multi-dimensional arrays are defined in the same manner as one dimensional arrays except that a separate pair of square brackets is required to each subscript.

Example: float matrix[20][20] (two dimensional)

Intx[10][10][5] (3-dimensional)

Initiating a two dimensional array we do as int x[3][4]={1,2,3,4,5,6,7,8,9,10,11,12}

Or

int x[3][4]={
{1,2,3,4};
{5,6,7,8};
{89,10,11,12};
}

NOTE: The size of the subscripts is not essential for initialization. For reading a two dimensional array we use two for-loop.

Example:

for(i=1;i<=2;++i)
for(j=1;j<=3;++j)
scanf("%f",&A[i][j]);

NOTE: If x[2][3]is a two dimensional array, the memory cells are identified with name x[0][0],x[0][1],x[0][2],x[1][0],x[1][1] and x[1][2].

Program for the addition of two matrices

```
#include<stdio.h>
 int main() {
   int i, j, mat1[10][10], mat2[10][10], mat3[10][10];
   int row1, col1, row2, col2;

   printf("\nEnter the number of Rows of Mat1 : ");
   scanf("%d", &row1);
   printf("\nEnter the number of Cols of Mat1 : ");
   scanf("%d", &col1);

   printf("\nEnter the number of Rows of Mat2 : ");
   scanf("%d", &row2);
   printf("\nEnter the number of Columns of Mat2 : ");
   scanf("%d", &col2);

   /* Before accepting the Elements Check if no of   rows and columns of both matrices is
equal */
   if (row1 != row2 || col1 != col2) {
      printf("\nOrder of two matrices is not same ");
      exit(0);
   }

   //Accept the Elements in Matrix 1
   for (i = 0; i < row1; i++) {
      for (j = 0; j < col1; j++) {
         printf("Enter the Element a[%d][%d] : ", i, j);
         scanf("%d", &mat1[i][j]);
      }
   }
```

```
  //Accept the Elements in Matrix 2
  for (i = 0; i < row2; i++)
    for (j = 0; j < col2; j++) {
      printf("Enter the Element b[%d][%d] : ", i, j);
      scanf("%d", &mat2[i][j]);
    }

  //Addition of two matrices
  for (i = 0; i < row1; i++)
    for (j = 0; j < col1; j++) {
      mat3[i][j] = mat1[i][j] + mat2[i][j];
    }

  //Print out the Resultant Matrix
  printf("\nThe Addition of two Matrices is : \n");
  for (i = 0; i < row1; i++) {
    for (j = 0; j < col1; j++) {
      printf("%d\t", mat3[i][j]);
    }
    printf("\n");
  }

  return (0);
}
```

Output:

```
Enter the number of Rows of Mat1 : 3
Enter the number of Columns of Mat1 : 3

Enter the number of Rows of Mat2 : 3
Enter the number of Columns of Mat2 : 3

Enter the Element a[0][0] : 1
Enter the Element a[0][1] : 2
Enter the Element a[0][2] : 3
Enter the Element a[1][0] : 2
Enter the Element a[1][1] : 1
Enter the Element a[1][2] : 1
Enter the Element a[2][0] : 1
Enter the Element a[2][1] : 2
Enter the Element a[2][2] : 1

Enter the Element b[0][0] : 1
Enter the Element b[0][1] : 2
Enter the Element b[0][2] : 3
Enter the Element b[1][0] : 2
Enter the Element b[1][1] : 1
```

Enter the Element b[1][2] : 1
Enter the Element b[2][0] : 1
Enter the Element b[2][1] : 2
Enter the Element b[2][2] : 1

The Addition of two Matrices is :
2 4 6
4 2 2
2 4 2

FUNCTIONS

Functions are programs.There are two types of functions- library functions and programmer written functions. We arefamiliarised with library functions and how they are accessed in a C program.

The advantage of function programs are many

1) A large program can be broken into a number of smaller modules.

2) If a set of instruction is frequently used in program and written as function program, it can be used in any program as library function.

Defining a function.

Generally a function is an independent program that carries out some specific well defined task. It is written after or before the main function. A function has two components-definition of the function and body of the function.

Generally it looks like

datatype function name(list of arguments with type)
{
Function body
return;
}

- **Return Type or data 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**.
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:** The function body contains a collection of statements that define what the function does.

If the function does not return any value to the calling point (where the function is accessed) .The syntax looks like

function name(list of arguments with type)
{
statements
return;

}

If a value is returned to the calling point, usually the return statement looks like **return(value).**In that case data type of the function is executed.

Note that if a function returns no value the keyword voidcan be used before the function name

Example:

```
(1) writecaption(char x[] );
{
printf("%s",x);
return;
}
(2) int maximum(int x, int y)
{
int z ;
z=(x>=y)? x : y ;
return(z);
}
(3) maximum(intx,int y)
{
int z;
z=(x>=y) ?x : y ;
printf("\n maximum =%d",z);
return ;
}
```

Note: In example (1) and (2) the function does not return anything.

Advantages of functions

1. It appeared in the main program several times, such that by making it a function, it can be written just once, and the several placeswhere it used to appear can be replaced with calls to the new function.
2. The main program was getting too big, so it could be made (presumably) smaller and more manageable by lopping part of it off and making it a function.
3. It does just one well-defined task, and does it well.
4. Its interface to the rest of the program is clean and narrow
5. Compilation of the program can be made easier.

Accessing a function

A function is accessed in the program (known as calling program)by specifying its name with optional list of argumentsenclosed in parenthesis. If arguments are not required then only with empty parenthesis.

The arguments should be of the same data type defined in the function definition.

Example:

```
1)  inta,b,y;
y=maximum(a,b);
2) char name[50] ;
```

writecaption(name);
3) arrange();
If a function is to be accessed in the main programit is to be defined and written before the main function after the preprocessor statements.
Example:

```
#include<stdio.h>
int maximum (intx,int y)
{
int z ;
z=(x>=y) ?x : y ;
return (z);
}
main( )
{
inta,b,c;
scanf("%d%d",&a,&b);
c=maximum(a,b);
printf("\n maximum number=%d",c);
}
```

Function prototype

It is a common practice that all the function programs are written after the main( ) function .when they are accessed in the main program, an error of prototype function is shown by the compiler. It means the computer has no reference about the programmer defined functions, as they are accessed before the definition .To overcome this, i.e to make the compiler aware that the declarationsof the function referred at the calling point follow, a declaration is done in the beginning of the program immediately after the preprocessor statements. Such a decleration of function is called prototype decleration and the corresponding functions are called function prototypes.

Example 1:

```
#include<stdio.h>
int maximum(intx,int y);
main( )
{
inta,b,c;
scanf("%d%d",&a,&b);
c=maximum(a,b);
printf("\n maximum number is : %d",c);
}
int maximum(int x, int y)
{
int z;
z=(x>=y) ?x : y ;
return(z);
}
```

Example 2:

```
#include<stdio.h>
voidint factorial(int m);
main( )
{
int n;
scanf("%d",&n);
factorial(n);
}
voidint factorial(int m)
{
inti,p=1;
for(i=1;i<=m;++i)
p*=i;
printf("\n factorial of %d is %d ",m,p);
return( );
}
```

Note: In the proPassing arguments to a functionThe values are passed to the function program through the arguments. When a value is passed to a function via an argument in the calling statement, the value is copied into the formal argument of the function (may have the same name of the actual argument of the calling function).This procedure of passing the value is called passing by value. Even if formal argument changes in the function program,
the value of the actual argument does not change.

Example:

```
#include<stdio.h>
void square (int x);
main( )
{
int x;
scanf("%d",&x);
square(x):
}
void square(int x)
{
x*=x ;
printf("\n the square is %d",x);
return;
}
```

In this program the value of x in the program is unaltered.

Function Arguments:

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

The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways that arguments can be passed to a function:

Call Type	Description
Call by value	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
Call by reference	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C uses **call by value** to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function and above mentioned example while calling max() function used the same method.

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable can not be accessed. There are three places where variables can be declared in C programming language:

1. Inside a function or a block which is called **local** variables,
2. Outside of all functions which is called **global** variables.
3. In the definition of function parameters which is called **formal** parameters.

Let us explain what are **local** and **global** variables and **formal** parameters.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables. Here all the variables a, b and c are local to main() function.

```
#include <stdio.h>

int main ()

{

  /* local variable declaration */

int a, b;

int c;

   /* actual initialization */

  a = 10;

  b = 20;

  c = a + b;
```

```
printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
return 0;
}
```

Global Variables

Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables:

```
#include <stdio.h>
 /* global variable declaration */
int g;
int main ()
{
  /* local variable declaration */
int a, b;
   /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;
printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
return 0;
}
```

A program can have same name for local and global variables but value of local variable inside a function will take preference. Following is an example:

```
#include <stdio.h>
 /* global variable declaration */
int g = 20;
int main ()
{
  /* local variable declaration */
```

```
int g = 10;
printf ("value of g = %d\n",  g);
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
value of g = 10
```

Recursion

It is the process of calling a function by itself ,until some specified condition is satisfied. It is used for repetitive computation (like finding factorial of a number) in which each action is stated in term of previous result

Example:

```
#include<stdio.h>
longint factorial(int n);
main( )
{
int n;
longint m;
scanf("%d",&n);
m=factorial(n);
printf("\n factorial is : %d", m);
}
longint factorial(int n)
{
if (n<=1)
return(1);
else
return(n*factorial(n-1));
}
```

In the program when n is passed the function, it repeatedly executes calling the same function for n, n-1, n-2,……..1.