PAGE NO.
DATE: 06/02/19

Ch-4

Arrays

★ Introduction:

Many application require the processing of multiple items that have common characteristics in such situation, it is convinient place the data item into an array where they will share the same name.

For eg: numerical data element represented by cell $x_1, x_2, \ldots x_n$ where x is common name / same name.

The individual data item can be character, integers, floating point etc, but they must all be of the same type under same name.

• Definition:

Arrays is a collection of similar data arrangement. Each array element is referred by specifying the array name followed by one or more subscript, with each subscript enclose in square bracket and subscript number is always non-negative integer.

eg.

for eg: x | | | | | | |

x[1] x[2] ..... x[n]

x is 'n' element - one dimensional array.

• Defining an Array:

Array are defined in much in the manner as ordinary variable, accept that each array name must accompanied a size specification that is number of element in square bracket. The general form of one dimensional array defination is -

```
data type array name (size);
int x (10);
char text (80); /* text is array name with
                   element or char type */
float text (5); /* percent is array name with
                   element of float type */
int R.No (size);
```

The above declaration is more convient to define an array size if term of symbol is constant. This makes is easier to modify the program that utilizes an array to the maximum array size.

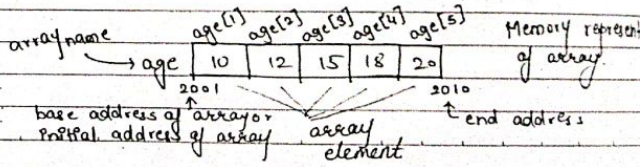
★ Array Initialization:

Array element are initialize in following way:

```
data type  Arrayname (size) = {value 1, value 2, ...};
int age [5] = {10, 12, 15, 18, 20};
```

that is equivalent to

```
int age [5];
age [1] = 10
age [2] = 12
age [3] = 15
age [4] = 18
age [5] = 20
```



PAGE NO.:
DATE: / /

```
char colour [3] = { 'R', 'E', 'D' };
    Colour [1] = 'R'
    Colour [2] = 'E'
    Colour [3] = 'D'
```

```
char country [5] = { 'I', 'N', 'D', 'I', 'A' }
```

I	N	D	I	A

★ Processing an Array:

Any operation such as assignment operation, comparison operation etc must be carried out an element by element basis. This is usually accomplish within a loop where each pass through the loop each loop to process one array element. The number of passes through the loop will equal the no. of array element to be processed.

```
/* Write a program to enter five element in array
and display it */
# include <stdio.h>
# include <conio.h>
void main ()
{
int i;
int list [5] = { 10, 5, 15, 12, 20 }; /* array declaration & initialization */
for (i=1; i<=5; i++)
{
printf ("\n %d", list [i]); /* display array element */
}
getch ();
}
```

PAGE NO.:
DATE: / /

07-02-19

★

```
/* Write a program to find greatest among the
list */
# include <stdio.h>
# include <conio.h>
void main ()
{
int i, max = 0;
int num (5);
printf (" Enter element for array");
for (i=1; i<=5; i++)
{
scanf ("%d", &num [i]); /* entering element in array */
}
for (i=1; i<=5; i++)
{
printf ("\n %d", num (i)]; /* printing element from array */
}
break;
/* checking for greatest number */
for (i=1; i<=5; i++)
{
if (num [i] >= max)
    max = num [i];
}
printf ("\n Largest no. is = %d", max);
getch ();
}
```

Output:

```
Enter the elements for array
45  56  70  13  50
Largest No is 70
```

PAGE NO.:
DATE: / /

* Passing Array to functions :

An entire array can be passed to a function as an argument. To pass an array to a function, the array name must appear by itself without bracket or subscripts, as an actual argument within the function called.

Similarly, the corresponding formal argument is return in the same manner in the function defination headline. The size of array is not specified within the formal argument declaration.

* The following program illustrate passing the array program function for calculating avg. of 10 nos.

```
#include <stdio.h>
#include <conio.h>
float average (int a, int num[]); /* fun° prototype eg*/.
void main ()
{
 int n, i;
 float avg;
 int list [10];
printf ("\n Enter array element");
 for (i=1 ; i<=10 ; i++)
{
 scanf ("%d", list [i]);
}
avg = average (n, list) ;    /* fun° call with array name */
printf ("\n areage of number = %f", avg);
 /* function defination */
float average (int a, int num);
 int sum = 0;
```

PAGE NO.:
DATE: / /

```
 float AVG.
{
 for (i=1 , i<=10 ; i++)
 {
  sum = sum + num(i);
 }
 AVG = sum/a;
 return (a);
}
```

eg:

08-02-19

Multidimensional Array :

Multidimensional arrays are define in same manner as one dimensional array except that a seperate pair of square bracket is require for each subscript thus two dimensional array will require two pairs of square bracket, three dimensional array will require of three (3) pairs of square bracket & so on.

The general form of multidimensional array is.

```
data type arrayname (expr1) (expr 2) .... (expr n);
```

eg:

```
float table (50) (50);  /* Two dimensional array of size 50by
double records (10) (5) ;
char name (5) (10);  /* Name is array of char type each n
                        contain 10 character & there are 6 nam
```

PAGE NO.:
DATE: / /

Initializing elements for two dimension.

eg: int value [3] [4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
value [1] [1] = 1, value [1] [2] = 2, value [1] [3] = 3, value [1] [4] = 4,
value [2] [1] = 5, value [2] [2] = 6, value [2] [3] = 7, value [2] [4] = 8.

Elements of two dimensional array will be assign by row i.e., the element of first row will be assign first, than the element of second row & so on. This is the natural order of assigning the element we can change this order by forming group of initial value enclose within braces as shown below

int value [3] [4] =

{{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};

Multidimensional or two dimensional array are process in same manner as one dimensional array i.e, on an element by element basis.

★ The following program illustrate the use of two dimensional array for addition of two matrix.

```
#include <stdio.h>
#include <conio.h>
void main ()
{
  int A [2] [2];
  int B [2] [2];
  int C [2] [2];
  int i, j;
```

PAGE NO.:
DATE: / /

```
  printf ("Enter element for matrix A");
  for (i=1; i<=2; i++)
  {
   for (j=1; j<=2; j++)
   {
    scanf ("%d", &A [i] [j]);
   }
  }
  printf ("\n Enter element for matrix B");
  for (i=1; i<=2; i++)
  {
   for (j=1; j<=2; j++)
   {
    scanf ("%d", &B [i] [j])
   }
  }
  printf ("\n Display matrix A and B");
  for (i=1; i<=2; i++)
  {
   for (j=1; j<=2; j++)
   {
    printf ("\t %d", A [i] [j]);
   }
   printf ("\n");
  }
  printf ("\n Display matrix A & B");
  for (i=1; i<=2; i++)
  {
   for (j=1; j<=2; j++)
   {
    print ("\t %d", B [i] [j]);
   }
```

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

/* addition of two matrix */
for (i=1; i<=2; i++)
{
for (j=1; j<=2; j++)
{
c[i][j]= A[i] [j] +B[i][j];
}
}

/* Print the result of Addition */
for (i=1; i<=2; i++)
{
printf("\t %d", c[i][j]);
}
printf ("\n");
getch ();
}
```

* Write program illustrate the use of two dimensional array for substraction of two matrix.

```
# include <stdio.h>
# include <conio.h>
void main()
{
int a [3][3];
int b [3][3];
int S[3][3];
int i, j;
printf (" Enter Elements for first 3*3);
for (i=0; i<=2; i++)
{
for (j=0; j<=2; j++)
{
scanf ("%d", &a[i][j]);
}
}
printf ("\n First Matrix: \n\n");
for (i=0, i<=2; i++)
{
for (j=0; j<=2; j++)
{
print f ("%d", a[i][j]);
}
print ("\n");
}
printf ("\n Enter Element for second 3*3 Matrix:\n");
for (i=0; i<=2; i++)
{
for (j=0; j<=2; j++)
{
scanf ("%d", &b[i][j]);
}
printf ("\n Second Matrix : \n\n");
for (i=0; i<=2; i++)
{
for (j=0; j<=2; j++)
{
printf ("%d", b[i][j]);
}
print f("\n");
```

```
    }
    for (i=0; i<=2; i++)
        for (j=0; j<=2; j++)
        {
        s[i][j] = a[i][j] - b[i][j];
        }
printf ("\n Difference b/w Matrices: \n\n");
    for (i=0, i<=2; i++)
    {
        for (j=0; j<=2; j++)
        {
        printf ("%3d", s[i][j]);
        }
    printf ("\n");
    }
    return 0;
}
```

13-02-19

Pointer:

Pointer is a variable that represent the location (rather than the value) of a data item, such as variable or any array element.

Pointers are used frequently in C as they have no. of useful applications such as (i) pointers can be used to pass information back & forth b/w a function & its reference point.

(ii) Pointer provide a way to return multiple data item from a function via function argument.

(iii) Pointer also permits references to other functions to be specified as argument to a given function.

(iv) Pointers are also used for accessing individual array elements.

(v) Pointers commonly used in multidimensional array.

Pointer Declaration:

As we know, every data item stored in memory occupies one or more contigeous memory cells. The no. of memory cells required to store data item depend on the type of data item. for eg: Integer data item requires 2 contigeous bytes, floating point no. may require 4 contigeous bytes, character data item require 1 byte etc.

Suppose, v is a variable that represent a particular data item, say int v=5; the compiler automatically assign a memory cell for that data item as shown below.

15/02/19

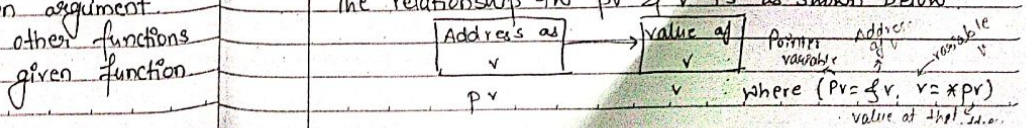
The element can be access if we know its location i.e., address. we can determine the location of variable the expression.

15-02-19.

'&'

that evaluates address of its operant. Let us assign the address of v to another variable pv as, pv = &v. This new variable is called pointer to v since it points to the location where 'v' is stored in memory.

It means that pv represents 'v's' address, not its value. Thus, pv is referred to as pointer variable. The relationship b/w pv & v is as shown below:



The data item represent by v i.e, stored in memory cell can be ~~ae~~ excess by the expression *pv, where * is a unary operator called indirectional operator that operates only on pointer variable.

The following program illustrate

```
# include <stdio.h>
# include <conio.h>
  void main ()
{
  int u=3;
  int v;
  int *pu;    /* pointer to an integer*/
  int *pv;    /* pointer to an integer */
  pu = &u;    /* assign address of u to pu*/
   v = *pu;   /* assign value of u to v */
  pv = &v;    /* assign value of v to pv */
printf ("\n u=%d, &u=%x pu=%x *pu=%d",
          u, &u, pu, *pu);
printf ("\n v=%d &v=%x pv=%x *pv=%d",
          v, &v, pv, *pv);
  getch ()
}
```

The indirection operator (*) can only act upon operant that are pointer

The following example shows how the pointer variable use for arithematic operation.

```
# include <stdio.h>
# include <conio.h>
  void main ()
{
  int u1, u2;
  int v=3;
  int *pv;      /* pv points to v*/
  u1 = 2* (v+5);   /* ordinary arithematic */
  pv = &v;
  u2 = 2* (*pv+5);  /* expression with pointer */
printf ("\n u1=%od u2 =%od", u1, u2)
  getch ();
}
```

output :

$u_1 = 16$

$u_2 = 16$

20-02-19

Pointer Declaration :

Pointer variable, like all other variables must be declared before they may be use in a program.

When a pointer variable is declared the variable name must be precedded by an astrix (*) sign.

This identifies that the variable is a pointer. Thus in general term the pointer declaration is

data type variable name;

where data type refers to the data type of pointer object and pointer variable is name of the variable.

In ex: ptr is a pointer variable name if it refers to the integer value.

Ex: float *pv;

* Passing pointer to a function:

Pointers are often pass to a function as arguments. These allows data items within the calling portion of the program to be access by the fun", altered within the fun" & then return to the calling portion of the program in altered form.

It is commonly used in passing arguments by reference instead of passing argument by value.

The following program illustrate the use of passing pointer to a function.

```
#include <stdio.h>
#include <conio.h>
void fun1 (int u, int v);
void fun2 (int *pu, int *pv);
main()
{
  int u=1;
  int v=3;
printf("\n Before calling fun1: u=%d v=%d", u,v);
fun1(u,v); /* fun" call */
printf("\n After calling fun1: u=%d v=%d", u,v);
printf("\n before calling fun2; u=%d v=%d", u,v);
fun2(&u, &v); /* fun" call with reference */
printf("\n After calling fun2; u=%d v=%d", u,v);
getch();
}

void fun1 (int u, int v) /* defn of fun" */
{
  u=0;
  v=0;
printf("\n with in fun" u=%d v=%d", u,v);
return;
}
void fun2 (int *u, int *v)
{
  u=0;
  v=0;
printf("\n with in fun" u=%d v=%d", u,v);
return;
}
```

Output:

```
Before calling fun1 u=1 v=3
with in fun" u=0 v=0
After calling fun1 u=1 v=3
before calling fun2 u=1 v=3
with in fun" u=0 v=0
After calling fun" u=0 v=0
```

21-02-19

* Pointer and one-dimensional Array :

As we know that array name is really a pointer to the first element in the array, therefore if x is one-dimensional array then the address of 1st array element can be expressed as either address of x[0] or just x.

Moreover, the address of 2nd array element can be written as address of x[1] or (x+1). & so on.

In general, the address of any element can be expressed as x[i] or (x+i).

Note that in the 2nd case we are dealing with very special and unusual type of expression i.e, (x+i) where x represent address i.e, name of array whose elements may be integer, float or char & 'i' is width of data type.

- The following program illustrate the relationship b/w array element their address & pointer.

```
# include <stdio.h>
# include <conio.h>
void main ()
{
static int x[5] = {10,11,12,13,14} ;
int i;
for (i=0; i<5; i++)
{
/* display array element */
printf ("\n i= %d  x[i]= %d  *(x+i)= %d ",
        i,  x[i]  *(x+i);
/* display corresponding address */
printf (" &x[i] = %o  x+i = %o", &x[i], (x+i));
}
getch();
}
```

output :

i=0	x[i] = 10	*(x+i) = 10	&x(i) = 3001	x+i = 3001
i=1	x[1] = 11	*(x+1) = 11	&x(1) = 3003	x+i = 3003
i=2	x(2) = 12	*(x+2) = 12	&x(2) = 3005	x+i = 3005
i=3	x(3) = 13	*(x+3) = 13	&x(3) = 3007	x+i = 3007
i=4	x(4) = 14	*(x+4) = 14	&x(4) = 3009	x+i = 3009