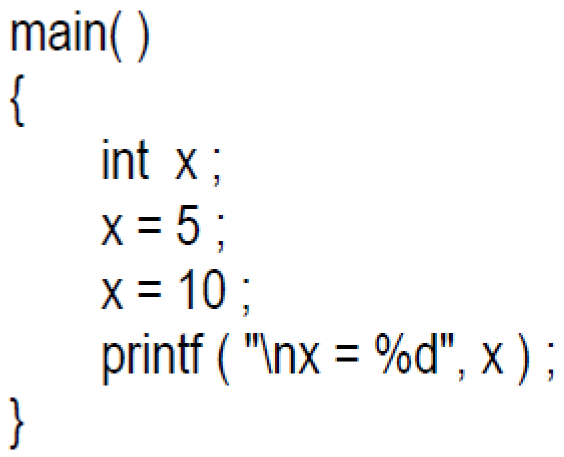
Arrays



```
main( )
{
       int  x ;
       x = 5 ;
       x = 10 ;
       printf ( "\nx = %d", x ) ;
}
```

- This program will print the value of **x** as 10. Why so?
- Because when a value 10 is assigned to **x**, the earlier value of **x**, i.e. 5, is lost.
- Thus, ordinary variables (the ones which we have used so far) are capable of holding only one value at a time (as in the above example).
- However, there are situations in which we would want to store more than one value at a time in a single variable.

- Suppose we wish to arrange the percentage marks obtained by 100 students in ascending order. In such a case we have two options to store these marks in memory:

(a) Construct 100 variables to store percentage marks obtained by 100 different students, i.e. each variable containing one student’s marks.

(b)Construct one variable (called array or subscripted variable) capable of storing or holding all the hundred values.

- The second alternative is better.
- A simple reason for this is, it would be much easier to handle one variable than handling 100 different variables.
- int a1,a2,a3,a4,…….,a100;
- Int a[100];

Array

- An array is a collective name given to a group of ‘similar quantities’.
- These similar quantities could be percentage marks of 100 students, or salaries of 300 employees, or ages of 50 employees.
- What is important is that the quantities must be ‘similar’.
- Each member in the group is referred to by its position in the group

- assume the following group of numbers, which represent percentage marks obtained by five students.

  per = { 48, 88, 34, 23, 96 }
- If we want to refer to the second number of the group, the usual notation used is per 2. Similarly, the fourth number of the group is referred as per 4.
- However, in C, the fourth number is referred as **per[3].**
- This is because in C the counting of elements begins with 0 and not with 1.
- Thus, in this example **per[3]** refers to 23 and
- **per[4]** refers to 96.
- In general, the notation would be **per[i]**, where, **i** can take a value 0, 1, 2, 3, or 4, depending on the position of the element being referred.
- Here **per** is the subscripted variable (array), whereas **i** is its subscript.

- An array is a collection of similar elements.
- These similar elements could be all **int**s, or all **float**s, or all **char**s, etc.
- Usually, the array of characters is called a ‘string’, whereas an array of **int**s or **float**s is called simply an array.

Program to find average marks obtained by a class of 30 students in a test.

```
main( )
{ int avg, sum = 0 ;
int i ;
int marks[30] ; /* array declaration */
for ( i = 0 ; i <= 29 ; i++ )
{ printf ( "\nEnter marks " ) ;
scanf ( "%d", &marks[i] ) ;
}
for ( i = 0 ; i <= 29 ; i++ )
sum = sum + marks[i] ; /* read data from an array*/
avg = sum / 30 ;
printf ( "\nAverage marks = %d", avg ) ;
```

```
main( )
{
 int avg, sum = 0 ;
int i , marks;
 printf ( "\nEnter marks " ) ;
scanf ( "%d", &marks ) ;
sum = sum + marks;
printf ( "\nAverage marks =
%d", avg ) ;
```

Array Declaration

```
int marks[30] ;
```

- Here, **int** specifies the type of the variable, just as it does with ordinary variables and the word **marks** specifies the name of the variable.
- The **[30]** however is new.
- The number 30 tells how many elements of the type **int** will be in our array.
- This number is often called the 'dimension' of the array.
- The bracket ([ ]) tells the compiler that we are dealing with an array

Accessing Elements of an Array

- This is done with subscript, the number in the brackets following the array name.
- This number specifies the element's position in the array.
- All the array elements are numbered, starting with 0.
- Thus, **marks[2]** is not the second element of the array, but the third.
- In our program we are using the variable **i** as a subscript to refer to various elements of the array.
- This variable can take different values and hence can refer to the different elements in the array in turn.

Entering Data into an Array

```
for ( i = 0 ; i <= 29 ; i++ )
{
  printf ( "\nEnter marks " ) ;
  scanf ( "%d", &marks[i] ) ;
}
```

- The **for** loop causes the process of asking for and receiving a student's marks from the user to be repeated 30 times.

- The first time through the loop, **i** has a value 0, so the **scanf( )** function will cause the value typed to be stored in the array element **marks[0],** the first element of the array.
- This process will be repeated until **i** becomes 29.

- There is no array element like **marks[30]**.

- In **scanf( )** function, we have used the "address of" operator (&) on the element **marks[i]** of the array, just as we have used it earlier on other variables **(&rate**, for example).

Reading Data from an Array

```
for ( i = 0 ; i <= 29 ; i++ )
        sum = sum + marks[i] ;
```

Review

- An array is a collection of similar elements.
- The first element in the array is numbered 0, so the last element is 1 less than the size of the array.
- An array is also known as a subscripted variable.
- Before using an array its type and dimension must be declared.
- However big an array its elements are always stored in contiguous memory locations.

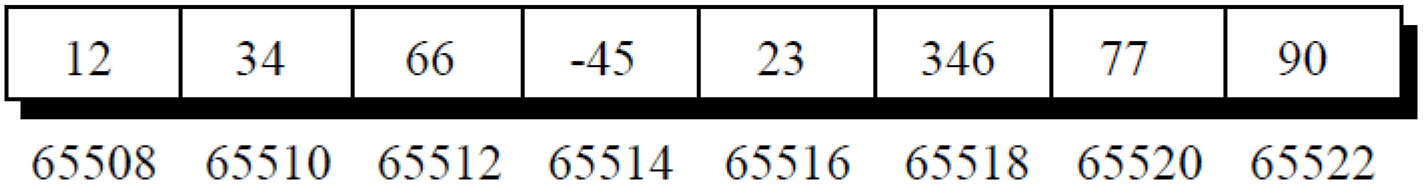
Array Initialisation

- int num[6] = { 2, 4, 12, 5, 45, 5 } ;
- int n[ ] = { 2, 4, 12, 5, 45, 5 } ;
- float press[ ] = { 12.3, 34.2 -23.4, -11.3 } ;

- Till the array elements are not given any specific values, they are supposed to contain garbage values.
- If the array is initialised where it is declared, mentioning the dimension of the array is optional as in the 2nd example above.

What happens in memory when we make this declaration?

- int arr[8] ;
- 16 bytes get immediately reserved in memory.
- 2 bytes each for the 8 integers
- Since the array is not being initialized, all eight values present in it would be garbage values



Bounds Checking

- In C there is no check to see if the subscript used for an array exceeds the size of the array.
- Data entered with a subscript exceeding the array size will simply be placed in memory outside the array; probably on top of other data, or on the program itself.
- This will lead to unpredictable results, and there will be no error message to warn you that you are going beyond the array size.

```
main( )
{
int num[40], i ;
for ( i = 0 ; i <= 100 ; i++ )
 num[i] = i ;
}
```

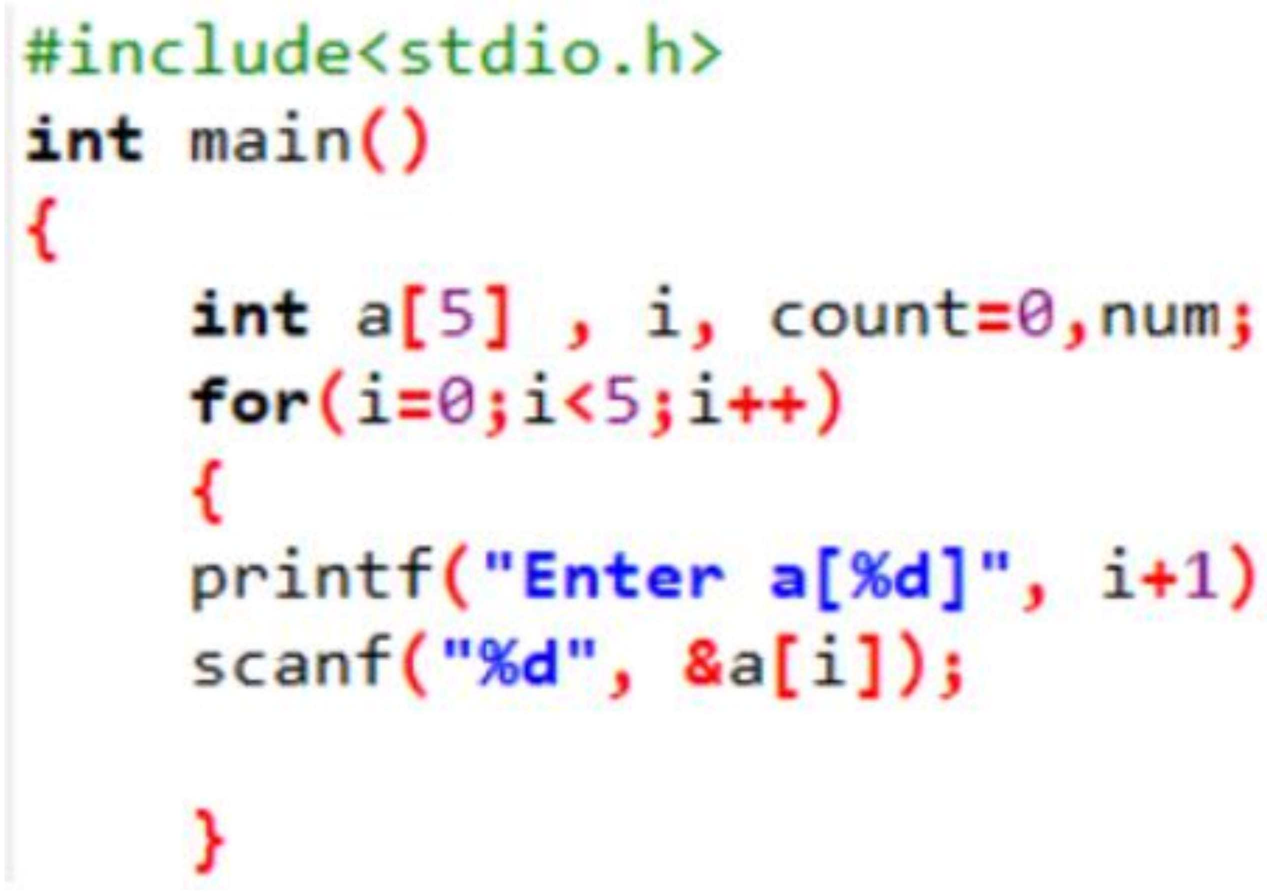
Program 1

Five numbers are entered from the keyboard into an array.

The number to be searched is entered through the keyboard by the user.

Write a program to find if the number to be searched is present in the array

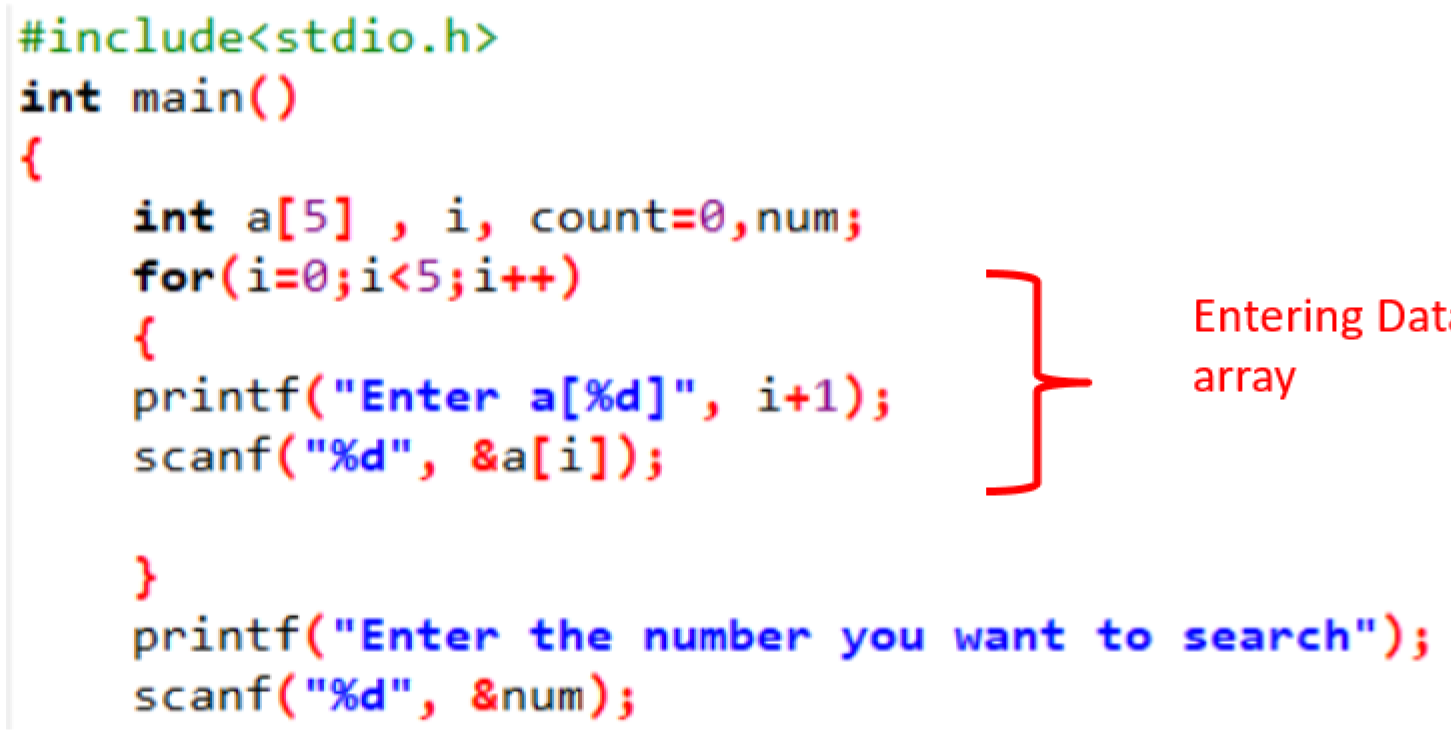
and if it is present, display the number of times it appears in the array.



```c
#include<stdio.h>
int main()
{
    int a[5] , i, count=0,num;
    for(i=0;i<5;i++)
    {
    printf("Enter a[%d]", i+1)
    scanf("%d", &a[i]);

    }
```

Entering Data in the array



```c
#include<stdio.h>
int main()
{
    int a[5] , i, count=0,num;
    for(i=0;i<5;i++)
    {
    printf("Enter a[%d]", i+1);
    scanf("%d", &a[i]);

    }
    printf("Enter the number you want to search");
    scanf("%d", &num);
```

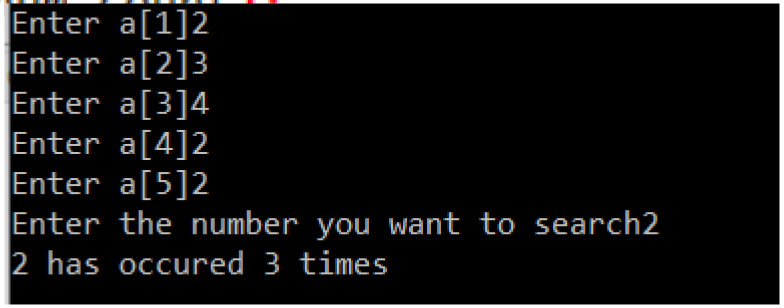
$\left.\vphantom{\begin{matrix}a\\b\\c\\d\end{matrix}}\right\}$ Entering Data in the array

```c
#include<stdio.h>
int main()
{
    int a[5] , i, count=0,num;
    for(i=0;i<5;i++)
    {
    printf("Enter a[%d]", i+1);
    scanf("%d", &a[i]);

    }
    printf("Enter the number you want to search");
    scanf("%d", &num);
    for(i=0;i<5;i++)
    {
        if(a[i]==num)
        count++;
    }
    printf("%d has occured %d times", num,count);
    return 0;
}
```

Entering Data in the array

Comparing num with every element of array



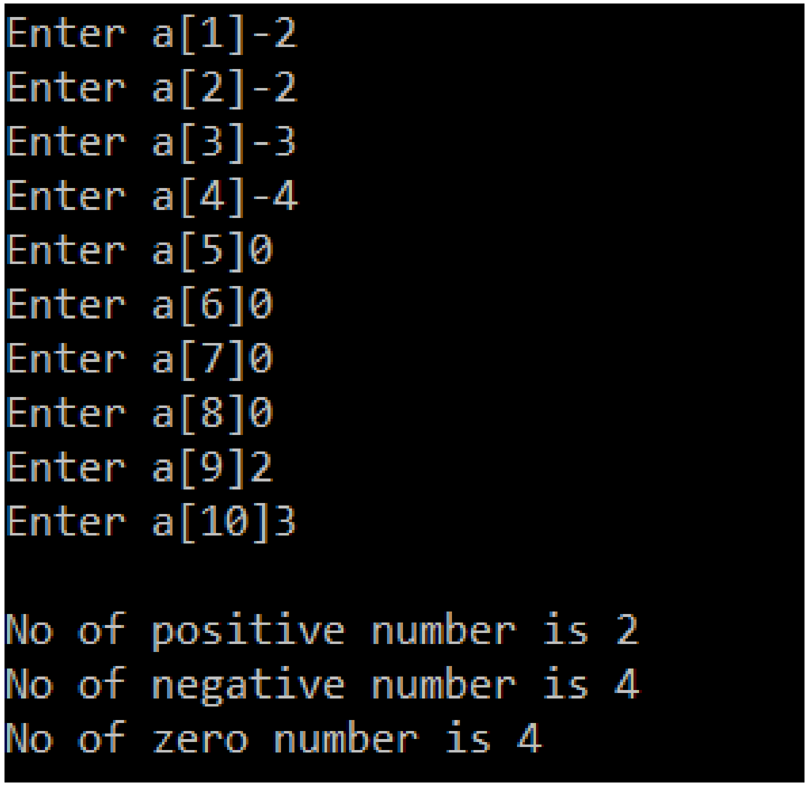
```
Enter a[1]2
Enter a[2]3
Enter a[3]4
Enter a[4]2
Enter a[5]2
Enter the number you want to search2
2 has occured 3 times
```

Program 2

- Ten numbers are entered from the keyboard into an array.
- Write a program to find out how many of them are positive, how many are negative, how many are even and how many odd.

```c
int main()
{
    int a[10] , i, countp=0,counto=0, countn=0;
    for(i=0;i<10;i++)
    {
    printf("Enter a[%d]", i+1);
    scanf("%d", &a[i]);

    }
    for(i=0;i<10;i++)
    {
        if(a[i]>0)
        countp++;
        else if(a[i]<0)
        countn++;
        else
        counto++;
    }
    printf("\nNo of positive number is %d", countp);
    printf("\nNo of negative number is %d", countn);
    printf("\nNo of zero number is %d", counto);
    return 0;
```



```
Enter a[1]-2
Enter a[2]-2
Enter a[3]-3
Enter a[4]-4
Enter a[5]0
Enter a[6]0
Enter a[7]0
Enter a[8]0
Enter a[9]2
Enter a[10]3

No of positive number is 2
No of negative number is 4
No of zero number is 4
```