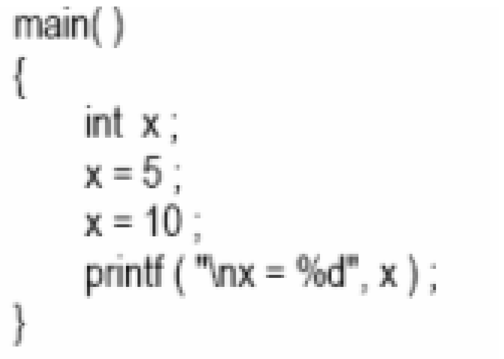
# ARRAY

Tanveer Ahmed Belal
Lecturer, Dept. of CSE

## Why Arrays



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

- No doubt, this program will print the value of **x** as 10
- Because when a value 10 is assigned to **x**, the earlier value of **x**, i.e. 5, is lost
- ordinary variables (the ones which we have used so far) are capable of holding only one value at a time

## Why Arrays

- 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:

  i) Construct 100 variables, each variable containing one student's marks.

  ii) Construct one variable capable of storing or holding all the hundred values.

## Why Arrays

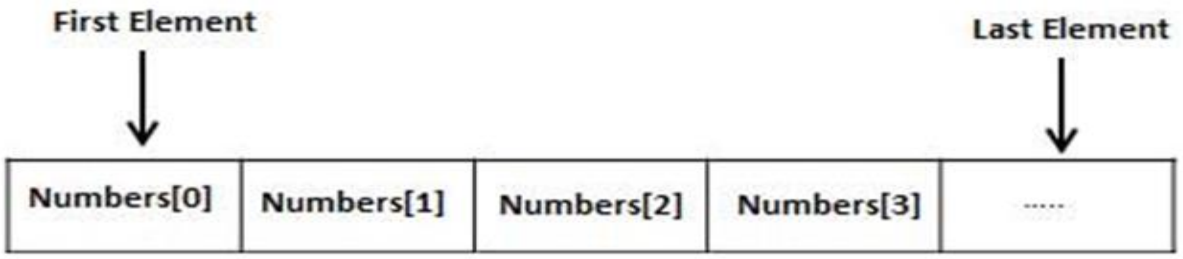
- the second alternative is better
- it would be much easier to handle one variable than handling 100 different variables
- Moreover, there are certain logics that cannot be dealt with, without the use of an array

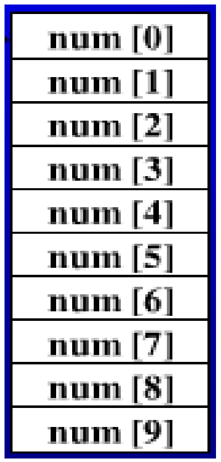
## Introducing array

- An array is a group of related data items that share a common name.
- C Array is a collection of variables to the same data type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- It is a group of memory locations related by the fact that they all have the same name.
- For example: int num[10];
- The individual values are called elements.

# Introducing array

- Array might be belonging to any of the data types
- Array size must be a constant value.
- Always, Contiguous (adjacent) memory locations are used to store array elements in memory.
- The lowest address corresponds to the first element and the highest address to the last element.





## Declaring array variables

- Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [ ] brackets for each dimension of the array.

  data_type array_name [array_size]

- This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type float, use this statement-

  float balance [10]

## Declaring array variables

- Type variable-name[size];
- Example:
  - `float height[50];`
  - `int group[10];`
  - `char name[10];`
- `int num[10];`
  - num[0] references the first element in the array.
  - num[9] references the first element in the array.

## Initialization of array during declaration

- int num[6] = { 2, 4, 12, 5, 45, 5 } ;
- The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].
- int num[ ] = { 2, 4, 12, 5, 45, 5 } ;
- If you omit the size of the array, an array just big enough to hold the initialization is created.
- float num[ ]={2.3, 5.6, 32.6};

## Initialization of array during declaration

- char name[]={‘a’,’u’,’v’,’i’,’\0’};
- Some exceptions:
  - int num[3]={0,0};
    - Auto insert 0 to the remaining index.
  - int num[3]={0,0,0,0};
    - If the initializing value is more than the size then it would cause a syntax error.

## The length of array

- Once array is created, its size is fixed. It cannot be changed.
- For example: int arr[10];
  - You cannot insert any number to arr[11], because arr[11] is not initialized.
  - int num[4]={2,3,4,5};
  - int num[4];
    - num[0]=2;
    - num[1]=3;
    - num[2]=4;
    - num[3]=5;

## The length of array

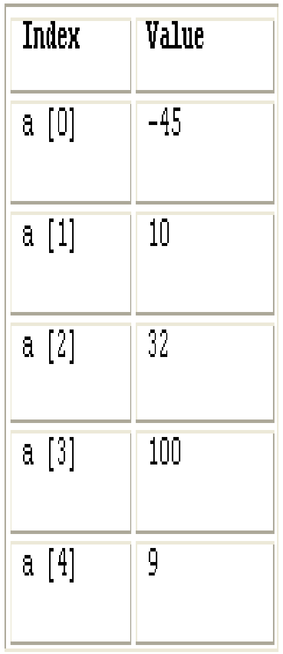
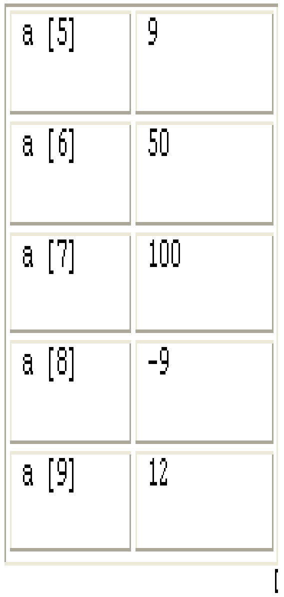
- Find the length of array

```
int array[8];
size_t = sizeof(array)/sizeof(array[0]);
```

- sizeof only works to find the length of the array if you apply it to the original array.
- Here size_t is an unsigned integral type which can represent the size of any object in bytes:

## One Dimensional Array

- C stores one dimensional array in one contiguous memory location with first element at the lower address
- an array named *a* of 10 elements can occupy the memory as follows-

| Index | Value |
|---|---|
| a [0] | -45 |
| a [1] | 10 |
| a [2] | 32 |
| a [3] | 100 |
| a [4] | 9 |
| a [5] | 9 |
| a [6] | 50 |
| a [7] | 100 |
| a [8] | -9 |
| a [9] | 12 |

## One Dimensional Array

```
main(){
        int a[10]={1,2,3,4,5,6,7,8,9,10},
        int i;
        printf ("Element \t Value");
        for (i=0;i<10;i++)
                printf ("%d \t%d", i, a [i]  );
}
```

- an array can also be initialized by following the declaration with an equal sign and a comma separated list of values within a pair of curly brace

## One Dimensional Array

```
main(){
        int a[10], i;
        for (i=0;i<10;i++)
                a [i] = 0;
        printf ("Element \t Value");
        for (i=0;i<10;i++)
                printf ("%d \t%d", i, a [i]  );
}
```

- a program that declares an array of 10 elements and initializes every element of that array with 9

## Simple program using Array

```
#include <stdio.h>
void main ()
 {
   int n[ 10 ];
   int i;
   for ( i = 0; i < 10; i++ )
 {
      scanf(“%d”,&n[i]);
}
   for (i = 0; i < 10; i++ )
 {
      printf("Element[%d] = %d\n", i, n[i] );
   }
}
```

## Simple program using Array

```
#include <stdio.h>
void main()
{
    int num[20], avg = 0,sum=0,x;
    for (x=0; x<=19; x++)
    {
        printf("enter the integer number %d\n", x);
        scanf("%d", &num[x]);
    }
    for (x=0; x<=19; x++)
    {
        sum = sum+num[x];
    }
    avg = sum/20;
    printf("%d", avg);
}
```

## C program to pass a single element of an array to function

```
#include <stdio.h>
void display(int age);
void main()
{
    int ageArray[] = { 2, 3, 4 };
    display(ageArray[2]);
}
void display(int age)
{
    printf("%d", age);
}
```

# Passing array to function

- When passing an array as a parameter like this

`void sendArray(int a[])` **means exactly the same as**
`void sendArray(int *a)`

- so you are modifying the values in main, as a result if we change a value in any function it will also change the main array value

****arrays cannot be passed by value****

So, there is no way to pass the array size except by using a second argument in your function that stores the array size

## Passing an entire one-dimensional array to a function

```
#include <stdio.h>
void showarray(int array[]);
void main()
{
       int n[] = { 1, 2, 3, 5, 7 };
       printf("Here's your array:");
       showarray(n);
}
void showarray(int array[])
{
        int x;
        for(x=0;x<5;x++)
        printf("%d\t",array[x]);
}
```

## Another Example

```
#include <stdio.h>
void showarray(int array[5]);
void main()
{
        int n[5],i;
        for(i=0;i<5;i++)
         scanf("%d",&n[i]);
        puts("Here's your array:");
        showarray(n);
}
void showarray(int array[])
{
         int x;
         for(x=0;x<5;x++)
        printf("%d\t",array[x]);
}
```

# Passing array to function with return type

```
#include <stdio.h>
double getSum(int arr[], int size);
void main () {
     int balance[5] = {10, 2, 3, 7, 5};
     double sum;
     sum = getSum( balance, 5 ) ;
     printf( "Summation value is: %f ", sum );
}
double getSum(int arr[], int size) {
     int i;
     double s = 0;s
     for (i = 0; i < size; ++i) {
        s += arr[i];
     }
     return s;
}
```