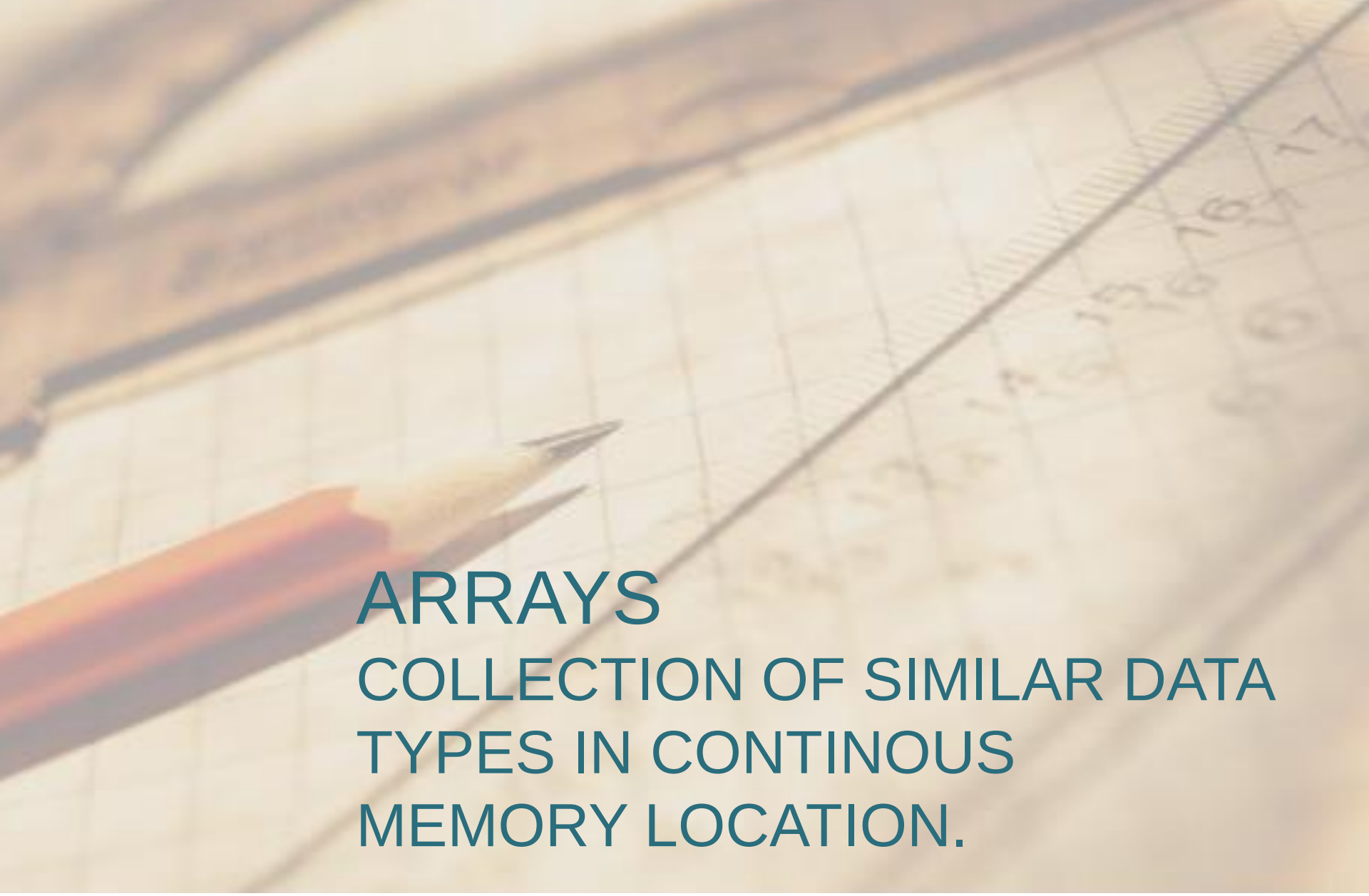
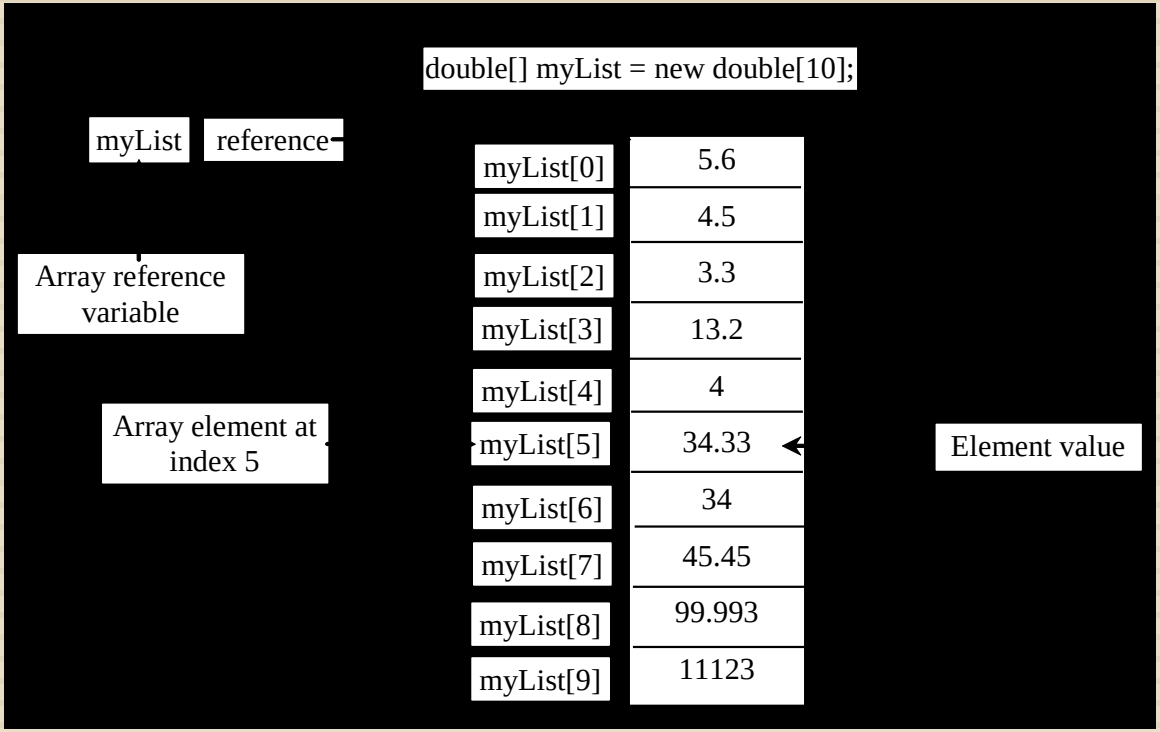


ARRAYS

COLLECTION OF SIMILAR DATA TYPES IN CONTINOUS MEMORY LOCATION.

Meaning

Array is a data structure that represents a collection of the same types of data



Arrays

- For understanding the arrays properly, let us consider the following program:
- main( )
- {
- int x ;
- x = 5 ;
- x = 10 ;
- printf ("\nx = %d", x) ;
- }

Cont.…

- No doubt, 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.
- For example, 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

Which Option You will Choosa

1. Construct 100 variables to store percentage marks obtained by 100 different students, i.e. each variable containing one student's marks.

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

I think second alternative is better . Why?

Array: Meaning

- 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.

A Simple Program Using Array

- Let us try to write a 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] ) ; /* store data in array */
}
for ( i = 0 ; i <= 29 ; i++ )
sum = sum + marks[i] ; /* read data from an array*/
avg = sum / 30 ;
printf ( "\nAverage marks = %d", avg ) ;
}
```

- There is a lot of new material in this program

Array Declaration

- To begin with, like other variables an array needs to be declared so that the compiler will know what kind of an array and how large an array we want. In our program we have done this with the statement:
- 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

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

Reading Data from an Array

- for (i = 0 ; i <= 29 ; i++)

{ sum = sum + marks[i] ;}

avg = sum / 30 ;

printf ("\nAverage marks = %d", avg) ;

Recap

- 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

- So far we have used arrays that did not have any values in them to begin with. We managed to store values in them during program execution. Let us now see how to initialize an array while

declaring it. Following are a few examples that demonstrate this.

- 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 } ;

QUESTIONS

- What would happen if you try to put so many values into an array when you initialize it that the size of the array is exceeded?
- 1. Nothing
- 2. Possible system malfunction
- 3. Error message from the compiler
- 4. Other data may be overwritten

QUESTION

- (c) What would happen if you put too few elements in an array when you initialize it?
- 1. nothing
- 2. possible system malfunction
- 3. error message from the compiler
- 4. unused elements will be filled with 0’s or garbage

QUESTION

- (d) What would happen if you assign a value to an element of an array whose subscript exceeds the size of the array?
- 1. the element will be set to 0
- 2. nothing, it’s done all the time
- 3. other data may be overwritten
- 4. error message from the compiler
- (e) When you pass an array as an argument to a function, what actually gets passed?
- 1. address of the array
- 2. values of the elements of the array
- 3. address of the first element of the array
- 4. number of elements of the array

Multidimensional arrays

- How to interpret a declaration like:

```
int d[2][4];
```

- This is an array with two elements:
  - Each element is an array of four `int` values
- The elements are laid out sequentially in memory, just like a one-dimensional array
  - Row-major order: the elements of the *rightmost* subscript are stored contiguously

PROGRAM

```
main( )
{
int stud[4][2] ;
int i, j ;
for ( i = 0 ; i <= 3 ; i++ )
{
printf ( "\n Enter roll no. and marks" ) ;
scanf ( "%d %d", &stud[i][0], &stud[i][1] ) ;
}
for ( i = 0 ; i <= 3 ; i++ )
printf ( "\n%d %d", stud[i][0], stud[i][1] ) ;
}
```

Initialising a 2-Dimensional Array

- int stud[4][2] = {
- { 1234, 56 },
- { 1212, 33 },
- { 1434, 80 },
- { 1312, 78 }
- } ;
- or even this would work...
- int stud[4][2] = { 1234, 56, 1212, 33, 1434, 80, 1312, 78 } ;
- of course with a corresponding loss in readability. It is important to remember that while initializing a 2-D array it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.

Declarations

- int arr[2][3] = { 12, 34, 23, 45, 56, 45 } ;
- int arr[ ][3] = { 12, 34, 23, 45, 56, 45 } ;
- are perfectly acceptable,
- whereas,
- int arr[2][ ] = { 12, 34, 23, 45, 56, 45 } ;
- int arr[ ][ ] = { 12, 34, 23, 45, 56, 45 } ;
- would never work.