Data Structure-Array

Kaustuv Bhattacharjee

University of Engineering & Management, Kolkata

Array-Introduction

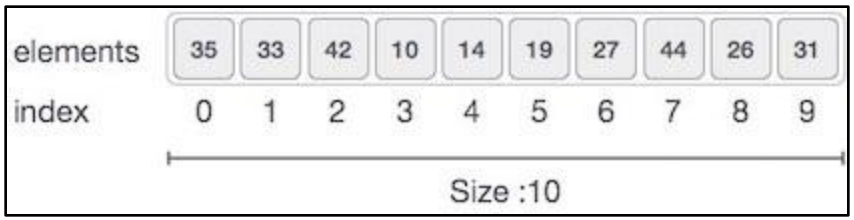
- A collection of fixed size similar data elements
- Data elements have same data type
- Elements are stored in consecutive memory locations
- Elements can be referenced by an index (called subscript), which is an ordinal number to identify and element of the array

Array-Declaration

- Like any other variables, array must be declared before being used
- Three things to be specified for array declaration: *Data type, Name, Size*
- Syntax:
  - type name[size];
- *Type* can be any valid data type (int, float, char, double etc)
- The number within square bracket indicates *size* of the array
- *Name* indicates the name of the array

Array-Declaration Contd…

- Example:
  - int name[10];
  - The above statement declares *name* is an array of 10 integer type elements.
  - Array index starts from 0.
  - Elements are stored as name[0], name[1], …, name[9].
  - 0,1,2,3,… are subscripts.



Array-Accessing Elements

- There is no single statement to read, access or print array elements
- To access Array elements, need to use loop; to execute the same statement with different access values
- Example:

```
int i, marks[10];
for (i=0,i<10,i++)
    scanf("The array element at index "%d" is:"%d, &i, &marks[i]);
```

The above code will access every data elements of the array and will set the value to the one entered by user.

Array-Address of array elements

- Array name refers to address of the first byte of the array
- The subscript / index represents offset from the beginning of the array to the element being referenced
- Example:
  - Address of marks[k] = BA(marks) + w (k-lower_bound)
  - Here, *marks* is the array, *k* is the index, *BA* is Base Address of the array *marks*, *w* is the size of one element in memory [if marks is *int* type array, then *w* is 2]

Array-Calculating Length

- Length of an Array is given by number of elements stored in it. General formula is:
  - Length=upper_bound – lower_bound + 1
  - Where *upper_bound* is the index of last element, *lower_bound* is index of first element in the array
- Example:

Let marks[5] be an array of integers such that

marks[0]=78,marks[1]=91,marks[2]=88,marks[3]=68,marks[4]=95

Show the memory representation of the array and calculate its length.

Array-Storing Values

- Three ways to store values in an array:
  - Initialize array during declaration
  - Input values for individual elements from console
  - Assign values to individual elements

Array-Storing Values Contd…

- Initialize array during declaration:
- Syntax: type array_name[size]={list of values};
  - **Ex 1:** int marks[5]={91, 65,47,88,93};
  - **Ex 2:** int marks[]={91, 65,47,88,93};
  - The above statement is legal. Compiler allocates enough space for all initialized elements
  - **Ex 3:** int marks[5]={87,98};
  - Here unassigned elements marks[2], marks[3], marks[4] are filled with zeros.

Array-Storing Values Contd…

- Input values for individual elements from console:
- Use a loop (*while/do-while* or *for* loop)
- Ex:

```
int i, marks[10];
for (i=0,i<10,i++)
    scanf(“The array element at index ”%d” is:”%d, &i, &marks[i]);
```

Array-Storing Values Contd…

- Assign values to individual elements:
- Using assignment (=) operator:
- Ex:
  - Marks[3]=100;
- One array can't be assigned to an array. This can be achieved in the following way:

```
int i, marks[5],marks2[5];
marks[5]={91, 65,47,88,93};
for (i=0,i<5,i++)
    marks2[i]=marks[i];
```

Array-Various Operations

- Traversing
- Insertion
- Deletion
- Searching (To be discussed later)
- Sorting (To be discussed later)
- Merging (To be discussed later)

Traversing an Array

- Traversing an array means the following: printing every element / counting number of elements/ perform any process on array elements
- <u>Steps/Algorithm:</u>

  Step 1: [Initialization] Set i= lower_bound

  Step 2: Repeat steps 3 and 4 while i<upper_bound

  Step 3: Apply Process to A[i]

  Step 4: set i = i + 1

  [loop ends]

  Step 5: Stop

Traversing an Array: Assignments

Assignments:

1. Write a program to read and display n numbers using an array.
2. Write a program to find the mean of n numbers using array.
3. Write a program to print the position of the smallest number of n numbers using array.
4. Write a program to find the second largest of n numbers using an array.
5. Write a program to find if an array of integers contains a duplicate number.

Inserting an element in an Array

- Let LA be a Linear Array (unordered) with N elements and K is a positive integer such that K<=N. Following is the algorithm where ITEM is inserted into the Kth position of LA
- Steps/Algorithm:

1. Start
2. Set J = N
3. Set N = N+1
4. Repeat steps 5 and 6 while J >= K
5. Set LA[J+1] = LA[J]
6. Set J = J-1
7. Set LA[K] = ITEM
8. Stop

Inserting an element in an Array- Assignments

Assignments:

1. Write a program to insert a number at a given location in an array
2. Write a program to insert a number in a sorted array (ascending/descending)

Deleting an element in an Array

- Let LA be a Linear Array with N elements and K is a positive integer such that K<=N. Following is the algorithm to delete an element available at the K^{th} position of LA.
- Steps/Algorithm:

1. Start
2. Set J = K
3. Repeat steps 4 and 5 while J < N
4. Set LA[J] = LA[J + 1]
5. Set J = J+1
6. Set N = N-1
7. Stop

Deleting an element in an Array- Assignments

Assignments:

1. Write a program to delete a number from a given location in an array
2. Write a program to delete a number from a sorted array (ascending/descending)

Passing Arrays to Functions

- Just like variables, array can also be passed to a function as an argument
- Can be achieved in 2 ways:
  - Passing as data (Call by Value)
  - Passing as address (call by reference)

Passing Arrays to Functions Contd…

- Passing as data (Call by Value)
  - Actual parameter is copied to formal parameters
  - Ex 1: Pass an individual Data element of an array

```
Calling function
Main()
{
        int num[]={1,2,3,4,5,};
        func(num[3]);
}
```

```
Called Function
 void func(int element)
{
    printf("%d", element);
 }
```

Passing Arrays to Functions Contd…

- Passing as data (Call by Value)
  - Actual parameter is copied to formal parameters
  - Ex 1: Pass an entire Array

```
Calling function
Main()
{
        int num[]={1,2,3,4,5,};
        func(num);
}
```

```
Called Function
 void func(int array1[5])
{
    int i;
   for (i=0;i<5,i++)
     printf("%d", array1[i]);}
```

Passing Arrays to Functions Contd…

- Passing as address (Call by reference)
  - Address of an array is passed
  - the function declaration should have a pointer as a parameter to receive the passed address

Calling Function

```
Main()
{
    int roll[]=(1,2,3,4,5};
    func(&roll[2]);
 }
```

Called Function

```
void func(int *element)
{
    printf(%d", *element);
}
```

Passing Arrays to Functions - Assignments

Assignments:

1. Write a program to read an array of n numbers and then find the smallest number: Use the concept of passing an array to a function
2. Write a program to interchange the largest and smallest number in an array: Use the concept of passing an array to a function

Arrays of Pointers

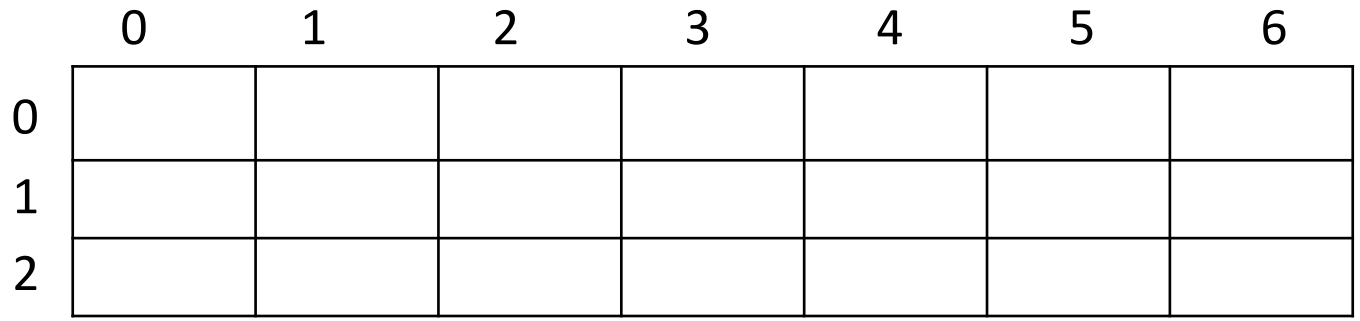
- An array of pointers can be declared as :
  - int *ptr[10];
  - Which declares an array of 10 pointers where each of the pointers point to an integer variable.
  - Ex:

```
int * ptr[5];
int p=1, q=2, r=3;
ptr[0]=&p;
ptr[1]=&q;
ptr[2]=&r;
printf("%d",*ptr[1]);
```

- In the above code, the output will be 2, as ptr[1] stores address of q, thus *ptr[1] will print the value of q that is 2.

Two-Dimensional Arrays

- The two-dimensional array can be defined as an array of arrays
- The 2D array is organized as matrices which can be represented as the collection of rows and columns
- Data stored in the form of grids or tables
- C compiler treats a two-dimensional array as an array of one-dimensional arrays



Two-Dimensional Arrays-Declaration

- Syntax of 2D array:
- data_type array_name[rows][columns];
- Ex: **int** marks [3][4]; 3 is number of rows, 4 is number of columns
- Pictorial form of 2D Array:

Rows / **Cols**	**Col 0**	**Col 1**	**Col 2**	**Col 3**
Row 0	marks [0][0]	marks [0][1]	marks [0][2]	marks [0][3]
Row 1	marks [1][0]	marks [1][1]	marks [1][2]	marks [1][3]
Row 2	marks [2][0]	marks [2][1]	marks [2][2]	marks [2][3]

Two-Dimensional Arrays-Initialization

- 2D array can be declared and defined as follows:
- **int** arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

  or as follows:
- **int** arr[4][3]={1,2,3,2,3,4,3,4,5,4,5,6};
- Initialization of a 2D array is done row by row.
- If the 2D array is completely initialized, size of the first dimension can be omitted.
- Ex: **int** arr[][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
- Entire 2D array can be initialized to zeros in the following way:
- int arr[4][3]={0};

Two-Dimensional Arrays-Accessing Elements

- To access 2D Array elements, need to use two loops
- The first for loop will scan each row in 2D array
- The second for loop will scan individual columns for every row in 2D array
- Example:

```
int i, j, marks[14][5];
for (i=0,i<10,i++)
    for (j=0,j<10,j++)
      scanf("The array elements are:", &marks[i][j]);
```

The above code will access every data elements of the 2D array and will set the value to the one entered by user

Two-Dimensional Arrays-Assignments

Assignments:

1. Write a program to print elements of a m X n 2D array
2. Write a program to read a 2D array marks which stores marks of five students in three subjects. Write a program to display the highest marks in each subject.
3. Write a program to transpose a m X n matrix.
4. Write a program to input two m X n ,matrices and then calculate the sum of their corresponding elements and store it in a third m X n matrix.
5. Write a program to calculate two m X n matrices.

Application of Arrays

1. Widely used to implement mathematical vectors, matrices and other kinds of rectangular tables
2. Many databases include 1D arrays whose elements are records
3. Arrays are used to implement other data structures like strings, stacks, queues, heaps and hash tables
4. Can be used for sorting elements (ascending/descending)

Multiple Choice Questions (MCQ)

1. If an array is declared as arr[] = {1,3,5,7,9}; then what is the value of sizeof(arr[3])?

(a) 1 (b) 2 (c) 3 (d) 8

2. If an array is declared as arr[] = {1,3,5,7,9}; then what is the value of arr[3]?

(a) 1 (b) 7(c) 9 (d) 5

3. If an array is declared as double arr[50]; how many bytes will be allocated to it?

(a) 50 (b) 100(c) 200 (d) 400

4. If an array is declared as int arr[50], how many elements can it hold?

(a) 49 (b) 50 (c) 51 (d) 0

5. If an array is declared as int arr[5][5], how many elements can it store?

(a) 5 (b) 25 (c) 10 (d) 0

6. Given an integer array arr[]; the ith element can be accessed by writing

(a) *(arr+i) (b) *(i + arr) (c) arr[i] (d) All of these

Multiple Choice Questions (MCQ)

Answers:

1. (b) 2. (b) 3. (d) 4. (b) 5. (b) 6. (d)