

**Table of Contents**

Contents

Lab 1: Arrays............................................................................................................................................5

1. Installation ......................................................................................................................................5

2. Introduction ....................................................................................................................................5

3. Activity Time boxing........................................................................................................................6

4. Lab Manual Lecture [Expected time = 20 minutes]...........................................................6

5. Objective .........................................................................................................................................6

6. Concept Map...................................................................................................................................6

Arrays ..................................................................................................................................................6

7.1 Update Operation ...................................................................................................................7

7.2 Search Operation ....................................................................................................................7

9. Evaluation criteria ...........................................................................................................................9

10. Further Reading ..........................................................................................................................9

10.1 Books.......................................................................................................................................9

7.7 Out comes .....................................................................................................................................9

––

**Lab Manual for Data Structure & Algorithm**

**Lab 1: Arrays**

**Lab 1: Arrays**

**1. Installation**

The IDE that we will be using in this course is *VS Code+* It is a free IDE for Windows that uses either MinGW

or TDM-GCC as underlying compiler.

To install please follow

**2. Introduction**

An array is used to process a collection of data all of which is of same type, such as a list of temperatures or list of names. 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.

For example, you want to store the score of 5 students, you can declare 5 variables, int student1, student2, student3, student4, student5;

Now what if the scenario changes and now we want to store the score of 100 students?

So instead of declaring individual variables, you declare one array variable such as **students** and use students [0], students [1], and ..., students [99] to represent individual variables.

**0 1 2 [n-1]**

**Index**

**Elements**

students [0] students [1] students [2] ………. students [n-1]

In this lab you will revise the basic working of arrays and pointers.

**3. Activity Time boxing**

Table 1: Activity Time Boxing

|  |  |  |  |
| --- | --- | --- | --- |
| **Task No.** | **Activity Name** | **Activity time** | **Total Time** |
|  | Lab Manual Lecture | 10 min |  |
|  | Examples | 5 min |  |
|  | Walkthrough Tasks | 10 min |  |
|  | Practice Tasks | 70 min |  |
|  | Tasks Evaluation | 10 min | 180 minutes |

4. Lab Manual Lecture [Expected time = 20 minutes]

**5. Objective**

1. Concepts of Data Structure

2. What is Array

3. CRUD operations on Arrays

4. Pointers

**6. Concept Map**

## What is a data structure?

It is a data organization, management and storage format that

enables [efficient access and modification](https://en.wikipedia.org/wiki/Algorithmic_efficiency).

Examples are arrays, Lists, Binary trees, Heaps.

Today we will discuss and revise arrays.

## Arrays

An array is a collection of data that holds fixed number of values of same type. For example:

int age[100];

Here, the age array can hold maximum of 100 elements of integer type.

The size and type of arrays cannot be changed after its declaration.

### Declaring array:

type arrayName [ arraySize];

int students [5]; **Initializing array:** students = {10, 8, 9, 5, 7};

**Sample Program**

#include <iostream>

using namespace std;

int main () {

int students [ 5]; // n is an array of 5 integers

// initialize elements of array for (int i = 0; i < 5; i++) {

cin>>students[i];

}

// output each array element's value

for (int j = 0; j < 10; j++) {

cout << “Student” <<j<< “scored” <<students[i] << “marks” <<endl;

}

return 0;

}

### 2.2. Array Operations

There are some basic operations that can be performed on array which are adding data element to array,

searching and updating particular data element in an array and deleting data element from array. Following are the detailed description of these operations

Following are the basic operations supported by an array.

* **Traverse** − print all the array elements one by one.
* **Insertion** − Adds an element at the given index.
* **Deletion** − Deletes an element at the given index.
* **Search** − Searches an element using the given index or by the value.
* **Update** − Updates an element at the given index.
  + 1. **Traverse Operation**

This operation is to traverse through the elements of an array.

1. Example

Following program traverses and prints the elements of an array:

#include <stdio.h>

main() {

int LA[] = {1,3,5,7,8};

int item = 10, k = 3, n = 5;

int i = 0, j = n;

cout<<”The original array elements are :\n";

for(i = 0; i<n; i++) {

cout<< LA[i];

}

}

* + 1. **Insertion Operation**

Insert operation is to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array.

Here, we see a practical implementation of insertion operation, where we add data at the end of the array Example

Following is the implementation of the above algorithm −

#include <stdio.h>

main() {

int LA[] = {1,3,5,7,8};

int item = 10, k = 3, n = 5;

int i = 0, j = n;

cout<<"The original array elements are :\n");

for(i = 0; i<n; i++) {

cout<<LA[i];

}

n = n + 1;

while( j >= k) {

LA[j+1] = LA[j];

j = j - 1;

}

LA[k] = item;

Cout<<”The array elements after insertion :\n";

for(i = 0; i<n; i++) {

cout<<LA[%d] = %d \n", i, LA[i];

}

* + 1. **Update Operation**

void Update Values ()

{

cout<<"Enter Index Number to Update Value:";

int index;

cin>>index;

cout<<"Enter the New Value For Index array [ "<<index<<" ] = ";

cin>>array[index];

cout<<"Array Updated... Successfully "<<endl;

}

* + 1. **Search Operation**

1. First take number of elements in array as input from user and store it in a variable **size**.

2. Using a loop, take input from user and store it in array (Let the name of the array be **inputArray**).

3. Ask user to enter element to be searched. Let it be num.

4. Now, using a for loop, traverse **inputArray** from index 0 to size-1 and compare **num** with every array element. If **num** is equal to any array element then print a message saying "Element found at index 4" otherwise print "Element Not Present".

* + 1. **Delete Operation**

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

### Algorithm

Consider **LA** is 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 Kth position of LA.

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

## Pointers

A **pointer** is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is −

type \*var-name;

Here, **type** is the pointer's base type; it must be a valid C++ type and **var-name** is the name of the pointer variable. The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration −

int \*ip; // pointer to an integer

double \*dp; // pointer to a double

float \*fp; // pointer to a float

char \*ch // pointer to character

### Use of Pointers

There are few important operations, which we will do with the pointers very frequently.

**(a)** We define a pointer variable.

**(b)** Assign the address of a variable to a pointer.

**(c)** Finally access the value at the address available in the pointer variable.

This is done by using unary operator \* that returns the value of the variable located at the address specified by its operand.

### How to use a pointer?

* Define a pointer variable
* Assigning the address of a variable to a pointer using unary operator (&) which returns the address of that variable.
* Accessing the value stored in the address using unary operator (\*) which returns the value of the variable located at the address specified by its operand.

The reason we associate data type to a pointer is **that it knows how many bytes the data is stored in**. When we increment a pointer, we increase the pointer by the size of data type to which it points.

## Reference operator (&) and Deference operator (\*)

Reference operator (&) as discussed above gives the address of a variable.

To get the value stored in the memory address, we use the dereference operator (\*).

**For example**: If a number variable is stored in the memory address **0x123**, and it contains a value **5**.

The **reference (&)** operator gives the value **0x123**, while the **dereference (\*)** operator gives the value **5**.

**Example:**

#include <iostream>

using namespace std;

int main() {

int \*pc, c;

c = 5;

cout << "Address of c (&c): " << &c << endl;

cout << "Value of c (c): " << c << endl << endl;

pc = &c; // Pointer pc holds the memory address of variable c

cout << "Address that pointer pc holds (pc): "<< pc << endl;

cout << "Content of the address pointer pc holds (\*pc): " << \*pc << endl << endl;

c = 11; // The content inside memory address &c is changed from 5 to 11.

cout << "Address pointer pc holds (pc): " << pc << endl;

cout << "Content of the address pointer pc holds (\*pc): " << \*pc << endl << endl;

\*pc = 2;

cout << "Address of c (&c): " << &c << endl;

cout << "Value of c (c): " << c << endl << endl;

return 0;

}

**Output**

Address of c (&c): 0x7fff5fbff80c

Value of c (c): 5

Address that pointer pc holds (pc): 0x7fff5fbff80c

Content of the address pointer pc holds (\*pc): 5

Address pointer pc holds (pc): 0x7fff5fbff80c

Content of the address pointer pc holds (\*pc): 11

Address of c (&c): 0x7fff5fbff80c

Value of c (c): 2

### NULL Pointers

A pointer that is assigned NULL is called a null pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries, including iostream.

### Void Pointer

A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be type casted to any type.

int a = 10;

char b = 'x';

void \*p = &a; // void pointer holds address of int 'a'

p = &b; // void pointer holds address of char 'b'.

### Pointer and arrays

While handling [**arrays**](https://beginnersbook.com/2017/08/cpp-arrays/) with pointers you need to take care few things. First and very important point to note regarding arrays is that the array name alone represents the base address of array so while assigning the address of array to pointer don’t use ampersand sign(&). Do it like this:  
**Correct:** Because arr represent the address of array.

p = arr;

**Incorrect:**

p = &arr;

#### Example: Traversing the array using Pointers

#include <iostream>

using namespace std;

int main(){

//Pointer declaration

int \*p;

//Array declaration

int arr[]={1, 2, 3, 4, 5, 6};

//Assignment

p = arr;

for(int i=0; i<6;i++){

    cout<<\*p<<endl;

//++ moves the pointer to next int position

    p++;

   }

return 0;

}

**Output:**

1

2

3

4

5

6

### How to increment pointer address and pointer’s value?

When we are accessing the value of a variable through pointer, sometimes we just need to increment or decrement the value of variable though it or we may need to move the pointer to next int position(just like we did above while working with arrays). The [**++ operator**](https://beginnersbook.com/2017/08/cpp-operators/) is used for this purpose. One of the example of ++ operator we have seen above where we traversed the array using pointer by incrementing the pointer value using ++ operator.

Lets see few more cases.

// Pointer moves to the next int position (as if it was an array)

p++;

// Pointer moves to the next int position (as if it was an array)

++p;

**Problem 1**: Write a function that takes an array of integers and returns the sum of its elements.

**Problem 2**: Given an array and a target element, count the occurrences of the target element in the array.

**Problem 3**: Write a function that returns all the even numbers from an array.

**Problem 4**: Given an array of integers, remove duplicates from the array and return the new array.

**Problem 5**: Write a function to check if an array is sorted in ascending order.

**Problem 6**: Rotate an array to the left by a given number of positions.

**Problem 7**: Write a function to find the second largest element in an array.

**Problem 8**: Find the sum of all elements at even indices in an array.

**Problem 9**: Given three sorted arrays, find the common elements in all three.

**Problem 10**: Given an array of n+1 integers where each integer is between 1 and n, there is exactly one duplicate. Find the duplicate number.