Data Structures

Introduction to Data Structure

- Data is a basic unit that any computing system centers around.

- What is data structure?

  Data Structure is an arrangement of data in computer's memory (or sometimes on a disk) so that we can retrieve it & manipulate it correctly and efficiently.

Definitions

- **Data type**

-Information itself has no meaning because it is just sequence of bytes.

- It is interpretation of bit pattern that gives it's meaning.

-For example 00100110 can be interpreted as number 38(binary), number 26(binary coded decimal) or the character '&'.

- A method of interpreting bit pattern is called as a data type.

- So data type is a kind of data that variable may hold in a programming language. For example int, float, char, double, class, …

Definitions

- **Data object** is a term that refers to set of elements. Such set may be finite or infinite.
- **Data structure** is a set of domains D, a set of functions F and a set of axioms A.
- A triple (D, F,A) denotes the data structure d.

Abstract Data Type

- ADT is a conceptual representation.
- ADT is a mathematical model together with various operations defined in that model.
- ADT is a way of looking at Data Structure focusing on what it does and not how it does.
- Data Structure is implementation of ADT.

Examples of Data Structure

- Arrays
- Stacks
- Queues
- Linked Lists
- Trees
- Graphs

Introduction to Arrays

- Arrays are defined as a finite ordered set of homogeneous elements.
- Finite means there is a specific number of elements in the array.
- Ordered means the elements of the array are arranged so that there is zeroth, first, second and so on elements.
- Homogeneous means all the elements in the array must be of the same type.
- E.g. An array may contain all integers or all characters but not integers and characters.
- Declaration in C++ `int a[ 100 ];`

Operations on Array

- The two basic operations that access an array are Extraction and Storing.
- The Extraction operation is a function that accepts an array a, and an index i, and returns an element of the array.
- This operation is denoted by the expression a[i];
- The storing operation accepts an array a, an index i and an element x.
  this operation is denoted by the expression

  a[i] = x;

Arrays Continued…

- The smallest index of an array's index is called it's lower bound. It is 0 in C++.
- The highest index is called it's upper bound.
- If Lower bound is "lower" and Upper bound is "upper" then number of elements in the array, called it's "range" is given by

  upper – lower + 1. → arr.lengths

E.g. In array a, the lower bound is 0 and the upper bound is 99, so the range is 100.

Arrays Continued..

- Arrays will always be stored in contiguous memory locations.
- You can have static as well as dynamic arrays, where the range of the array will be decided at run time.

Single Dimension Array Implementation

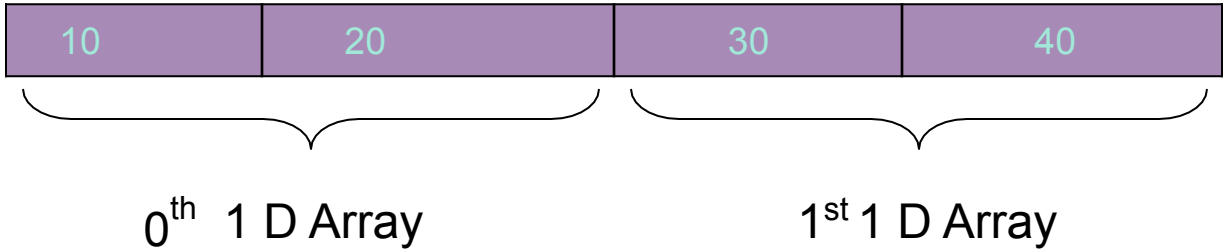
```
Int arr[], i;
arr=new int[10]
for( i = 0; i < 10; i++ )  arr[i] = i;

for ( i = 0; i < 10; i++ )

  cout<< i ;
```

Two Dimensional Array

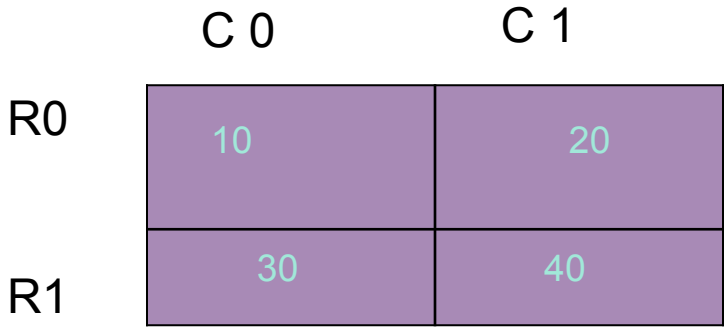
- Two dimensional array can be considered as an array of 1-D array.
- Declaration int arr[ ][ ]=new int[2][2];
- This defines a new array containing two elements. Each of these elements is itself an array containing 2 integers.
- Each element of this array is accessed using two indices: row number and the column number.

2-D Array Representation

- Physical:



- Logical:



Using 2D Array

```
Int arr[][], i , j;

For( i = 0; i < 2; i++ )

  for( j = 0; j < 2; j++ )

            arr[ i ][ j ] = I + j;


For( I = 0; i < 2; i++ )

  for( j = 0; j < 2; j++ )

            Sysout(arr[ i ][ i ];
```

Advantages of Arrays

- Simple and easy to understand.
- Contiguous allocation.
- Fast retrieval because of indexed nature.
- No need for the user to be worried about allocation and deallocation of arrays.

Disadvantages of Array

- If you need m elements out of n locations defined. n-m locations are unnecessarily wasted if n>m
- You can not have more than n elements. (I.e static allocation of memory.)
- large number of data movements in case of insertion & deletion, which leads to more overheads.