DATA STRUCTURES AND ALGORITHMS

SLIDE 2

Data Structures; an Introduction

The computer can be used for among many things:

- Scientific Calculations
- Information Management System for an enterprise
- Real-time control system for an assembly line

Introduction

- In performing these operations, to use the computer effectively one needs to:

a) Obtain the needed data and their relations

b) Store the data into the computer according to their relations

c) Design algorithms to solve the problem

Introduction

- The tasks of programming and data processing require *efficient algorithms* for accessing the data both in main memory and on secondary storage devices.

- This efficiency is directly linked to the structure of the data being processed. A data item that can be effectively linked to other data items takes on meaning that transcends its individual value.

- A data structure is a way of organizing data that considers not only the items stored but also their relationship to each other.

Introduction

- The way data is accessed is a function of how it is organized, thus a general understanding of data structures is essential to developing efficient algorithms.

- A clear notion of the relative advantages and disadvantages of each technique is obviously crucial to those designing systems.

Introduction

- The ability to make correct decisions regarding which data structures to use typically involve the following general issues:

a) The efficiency of the program with respect to its **run time**. Does it perform the optimal number of operations to achieve its goal?

b) The efficiency of a program with respect to **its utilization of main memory and secondary storage** devices. Does it consume such resources in a fashion that makes its use impractical?

c) The **developmental costs** of a program or system. Could a different approach to the problem significantly reduce the total person-hours invested in it?

Introduction

- Thorough knowledge of a programming language is not a sufficient base upon which to make these decisions.
- The study of data structures will expose you to a vast collection of tried and proven methods used in designing efficient programs.
- One data structure sacrifices memory compactness for speed; another utilizes memory efficiently but results in a slow run time.
- For each positive there is seemingly a corresponding negative.

Basic definitions

i. **Data structures -** In computer science, this is a way of storing data in a computer so that it can be used efficiently. Often a carefully chosen data structure will allow a more efficient algorithm to be used.

ii. An **abstract data type** (**ADT**) - is a specification of a data structure and the set of operations that can be performed on it. E.g a queue

iii. **Abstraction -** This is the separation between what a data structure represents and what an algorithm accomplishes, from the implementation details of how things are actually carried out. i.e., hiding the unnecessary details

iv. **Data Abstraction -** Hiding of the representational details

v. **Data Types - a data type**, or **type** is a classification of data. It indicates a set of values that have the same general meaning or intended purpose, e.g primitive types(integers,strings),records,classes

Atomic/Simple Data Types

- These are data types that are defined without imposing any structure on their values (have no internal structural relationship)

- Examples of atomic types.

  - Boolean,
  - Integer
  - characters

Structured Data Types

- A structured data type has a definition that imposes structure upon its values.
- In many structured data types, there is an *internal structural relationship*, or organization, that holds between the components.
- For example, if we think of an array as a structured type, with each position in the array being a component, then there is a structural relationship of `*followed by*': we say that component i is followed by component $i+1$.
- Many structured data types do have an *internal structural relationship*, and these can be classified according to the properties of this relationship.

Structured Data Types : Arrays

- An Arrays is a **physically sequential**, fixed size collection of homogeneous objects. In addition, objects with the structure have the random access property.

I. An array consists of a set of cells, which are contiguous

II. The array is physically sequential in that the data objects are stored in consecutive memory locations.

III. The array has a fixed size in that its size can neither be increased nor reduced though the number of items it contains can vary

IV. The array is homogeneous in that they are made up of objects that are all the same type.

.

V Each cell in an array has an index which uniquely identifies the cell in an array.

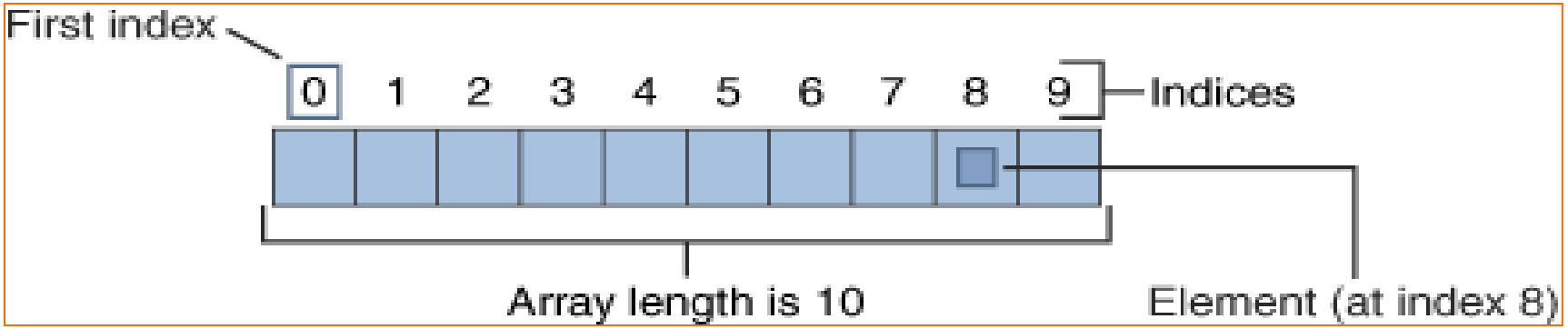
VI An array has a name which identifies the entire file

Declaring an Array

- *Data_type Array_name[Size]*
- *Data_type ARray_name[ROW][COL]*
- *Data_type Array_name[X][Y][Z]*

Eg float NUM[10]

Arrays



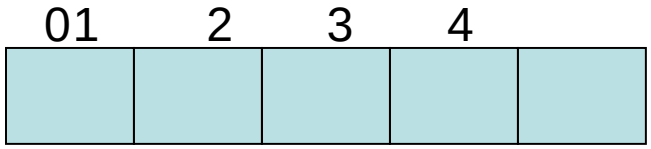
- An array type is appropriate for representing an abstract data type when the following conditions are satisfied:

1) *The data objects in the abstract data type are composed of homogeneous objects*
2) *The solution requires the representation of a fixed, predetermined number of objects*

Arrays

Examples: Storing values into an array and getting the greatest value using C

```
int Numbers[5];
```




```
for (int i=0;i<5;i++)
{
      cout<<("Enter Next Number:- ");
      cin>>num[i];
}
```

8	2	22	56	11

```
int big=Numbers[0];
for (i=0;i<5;i++)
{
    if(Numbers[i]>big)
    big=Numbers[i];
 }
```

Big=8	Big=8	Big=22	Big=56	Big=56

Arrays

```
// Example: Outputting the elements of an array
#include <iostream>
using namespace std;
int main ()
{
int billy [] = {16, 2, 77, 40, 12071};
int n;
    for ( n=0 ; n<5 ; n++ )
    {
          cout<<billy[i];
    }
         return 0;
}
```

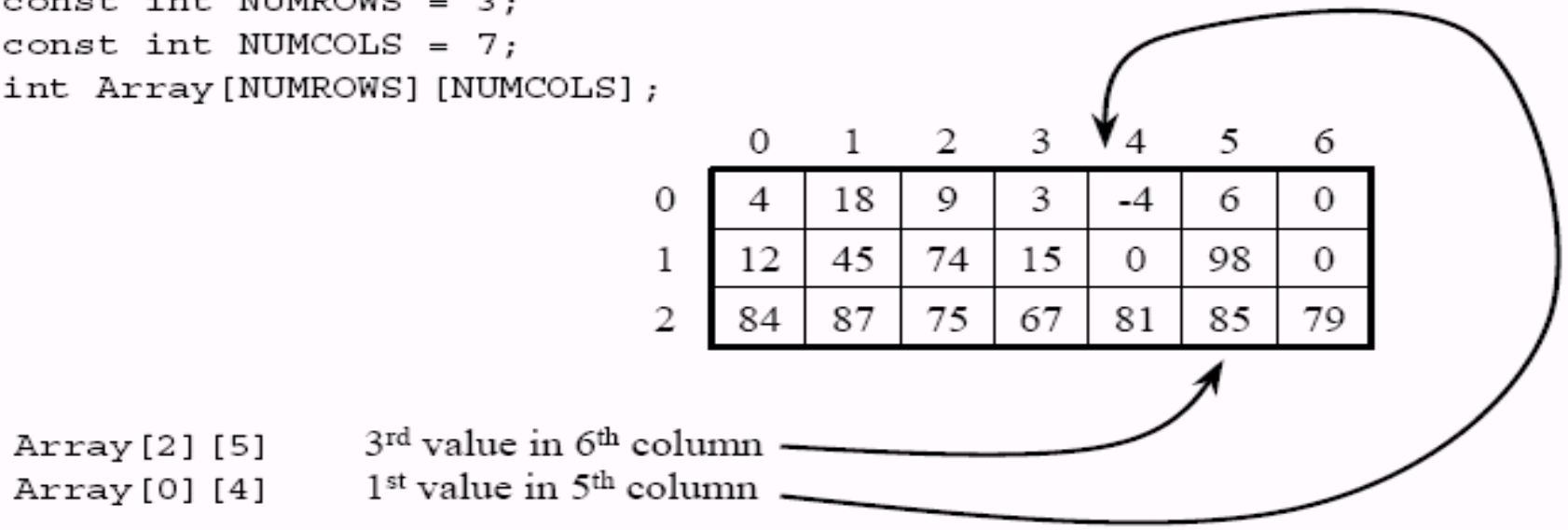
Arrays

```
#include<iostream>
using namespace std ;
int main()
{
int Numbers[10], i, big;
cout<<"We are dealing with arrays"<<"\
    n";
cout<<"*** Assign values ***";

    for (int i=0;i<10;i++)
    {    cout<< "Enter Number in Cell:-
    "<<i<<"\n";
          cin>>Numbers[i];
   }
```

```
cout<<"Elements Entered\n";

        for ( i=0;i<10;i++)
        {  cout<<Numbers[i]<<",";
        }
//Get the biggest element

big=Numbers[0];

for (i=0;i<10;i++)
{
        if(Numbers[i]>big)
        big=Numbers[i];
}

cout<< "The Bigest number is  :"<< big;
}
```

Multidimensional arrays

```
const int NUMROWS = 3;
const int NUMCOLS = 7;
int Array[NUMROWS][NUMCOLS];
```



- Multidimensional arrays can be described as arrays of arrays. For example, a two-dimensional array consists of a certain number of rows and columns:
- A one-dimensional array is usually processed via a ‘for’ loop. Similarly, a two-dimensional array may be processed with a nested for loop:

What does the piece of code below do?

```
for (int Row = 0; Row < NUMROWS; Row++)
{
  for (int Col = 0; Col < NUMCOLS; Col++)
  {
     Array[Row][Col] = 0;
  }
}
```

ARRAY ADVANTAGES

- *Fast execution speed*
- *Simple to occupy*
- *Occupy less space*

Arrays:Disadvantages

1) The size of the array is fixed —

   Most often this size is specified at compile time with a simple declaration such as in the example above. With a little extra effort, the size of the array can be deferred until the array is created at runtime, but after that it remains fixed.

2) Because of (1), the most convenient thing for programmers to do is to allocate arrays which seem "large enough" (e.g. the 100 items). Although convenient, this strategy has two disadvantages:

   - most of the time there are just 20 or 30 elements in the array and 70% the space in the array really is wasted.

   - If the program ever needs process more than 100 scores, the code breaks.

3) *Inserting new elements at the front or somewhere at the middle is potentially expensive because existing elements need to be shifted over to make room.*

4) *Inefficient in terms of memory utilization. This is because it requires contiguous memory space and available memory is normally fragmented*

Exercise 1

A matrix can be represented using a two dimensional array. Create a C++ program that does the following

i. *creates a three by three matrix.*
ii. *Allows a user to enter values into the elements of the matrix*
iii. *Outputs the elements of the matrix*
iv. *Sums all the elements of the matrix and outputs the result*

END.

WAIRAGU G.R