Southern Technical University
Technical Institute / Qurna
Dep. of Computer Systems Techniques

Second class
Subject : Data Structures
Lecturer : Israa Mahmood Hayder
Lecture no.4

هياكل البيانات المركبة

(Compound Data Structures)

- الاسبوع الرابع-

الرابع - الخامس	* هياكل البيانات المركبة Compound Data Structures . - المصفوفات Arrays . - تمثيل المصفوفات. - تمثيل المصفوفة الأحادية في الذاكرة. - تمثيل المصفوفة الثنائية في الذاكرة. - طريقة الصفوف. - طريقة الأعمدة.

B// Rationale (مبررات الوحدة) :-

Array is a very basic data structure representing a group of similar elements, accessed by index. Array data structure can be effectively stored inside the computer and provides fast access to the all its elements. Let us see an advantages and drawbacks of the arrays.

C// Central Ideas (الفكرة المركزية):-

- Introduction to compound data structures:
- Definition of linear Data Structure
- Declaring the arrays in C++
- Representation of one dimensional arrays

D// Objectives (أهداف الوحدة):-

After studying this unit, the student will be able to:-

- Define Linear Data Structure

- Learn how to declare and use the arrays in C++
- Represent the one dimensional array in memory

Real Memory Addresses

Memory addresses are represented so far throughout this chapter as a decimal value, such as "423" In reality, memory addresses are a 32-bit or 64-bit number, depending on the computer's operating system, and are represented as a hexadecimal value.

Hexadecimal is a numbering system similar to the decimal and binary numbering systems. That is, hexadecimal values are used to count and they are used in arithmetic. The hexadecimal numbering system has 16 digits from 0 through 9 and A through F, which represents 10 through 15. Here is how memory address 258,425,506 is represented in hexadecimal notation 0x0F6742A2.

Since each byte of memory has its own memory address, you might assume a short has two memory addresses because it uses 2 bytes of memory. That's not the case. The computer uses the memory address of the first byte to reference any abstract data type that reserves multiple bytes of memory. Let's say that space was reserved in memory for a short abstract data type. Two memory locations are reserved, memory addresses 400 and 401. However, only memory address 400 is used to reference the short .

Declaring an Array

The way to declare an array depends on the programming language used to write your program. In Java, there are two techniques for declaring an array. You can declare and initialize an array either where memory is allocated at compile time or where memory is dynamically allocated at runtime. *Allocation* is another way of saying reserving memory.

Let's begin by declaring an array where memory is reserved when you compile your program. This technique is similar in Java, C, and C++, except in Java you must initialize the array when the array is declared. There are four components of a statement that declares an array. These components are a data type, an array name, the total number of array element to create, and a semicolon (;). The semicolon tells the computer that the preceding is a statement. Here's the declaration statement in C and C++:

```
int grades[10];
```

In Java, you must initialize the array when the array is declared as shown here. The size of the array is automatically determined by counting the number of values within the braces. Therefore, there isn't any need to place the size of the array within the square brackets:

```
int[] grades = {0,0,0,0,0,0,0,0,0,0};
```

The *data type* is a keyword that tells the computer the amount of memory to reserve for each element of the array. In this example, the computer is told to reserve enough memory to store an integer for each array element.

The *array name* is the name you use within a program to reference an array element. The array name in this example is grades. The number within the square brackets is the total number of elements that will be in the array. The previous statements tell the computer to create an array of 10 elements.

Declaring a Multidimensional Array

A multidimensional array is declared similar to the way you declare a one-dimensional array except you specify the number of elements in both dimensions. For example, the multidimensional array declared as follows in C or C++:

```
int grades[3][4];
```

You assign a value to an element of a multidimensional array with an assignment statement similar to the assignment statement that assigns a value to a single-dimensional array, as shown here:

```
grades[0][0] = 1001;
```

You must specify the index for both dimensions. In this example, the integer 1001, which is a student ID, is assigned to the first element of the first set of elements in the grades array.

(Compound D.S):-

1) array 2) stacks

1) **Linear D.S :-**

Is a list consisting of variable number of elements to which the insertion and deletion can be made.

Storage structure for One Dimentional Array (Vector):-

The simplest data structure that use of computed address to locate it's elements is the one dimentional array (or vector).

is an array of n of elements consist of n of sequential elements in the memory, the size of the array and the required locations used to save in the memory is constant.

To find the location of any element use the following equation;-

Loc (A(I)) = Loc + C * (I-1)

Such that

A(I) is the required element

Loc is the starting address to store

C is the number of bytes for each element.

Ex: In the array B(12), Assume 2 bytes of storage are required to store the elements, Assuming that storage for the array begins at byte 300 in memory, give the actual address of the elements B[7] and B[10].

Loc (B[7]) = 300 +2* (7-1) = 312

Loc (B[10]) = 300 + 2* (10-1) = 318

One Dimentional Arrays in C++:

An array is a way to reference a series of memory locations using the same name. An array element is similar of name.

grade[0]
grade[1]
grade[2]

all elements are of the same type

int grade[8] ;
grade[3] = 90 ;

Ex1 :- adding elements of an array :-

```
sum =0 ;
for (int i=0 , i<100 , i++)
    sum = sum + grade[i] ;
```

Ex2 :-

```
char letters [3];
letters [0] = 'C' ;
letters [1] = 'B' ;
letters [2] = 'A' ;
```

Ex2 :-

Array of pointers (pointer is a variable that contains a memory address of another variable).

Array of pointers is identical to a pointer variable except each array element contains a memory address.

```
char  *ptr[3] ;
for (int i=0 , i<3 , i++)
    ptr[i] = & letters[i] ;
```

Ex:-

```
# include <iostream.h>
int main( )
{ int A[9]={3,-1,0,10,7,4,11,15,2};
  float B[9] = {1.5,1.0,5.1,12.5,7.0,4.1,1.1,2.0,2.8};
  double C[9] ;
  Int i ;
   cout<< "A={";
   for (i=0,i<9;i++)
         cout<<"\t"<<A[i];
```

طبع عناصر A

```
 cout <<"}\n";

cout<<"B={";
for (i=0,i<9;i++)
        cout<<"\t"<<B[i];
 cout <<"}\n";

cout << "\n C=A+B\n";
 cout<< "C={";
 for (i=0,i<9;i++)
     {
        C[i]=A[i]+B[i];
        cout<<"\t"<<C[i];
       }
 cout <<"}\n";

cout<<"\n C=A*B\n";
 cout<< "C={";
 for (i=0,i<9;i++)
     {
        C[i]=A[i]*B[i];
        cout<<"\t"<<C[i];
       }
 cout <<"}\n";

 cout<<"\n C=A-B\n";
 cout<< "C={";
 for (i=0,i<9;i++)
     {
        C[i]=A[i]-B[i];
        cout<<"\t"<<C[i];
       }
 cout <<"}\n";
return 0;     }
```

طبع عناصر B

طبع عناصر B + A

طبع عناصر B * A

طبع عناصر B - A

Quiz1:

Write program to read array a[20] if integers and print the number of positive and negative values.

Quiz2

In the array B(12), Assume 3 bytes of storage are required to store the elements, Assuming that storage for the array begins at byte 500 in memory, give the actual address of the elements B[6]

References:

1 Data Structures Demystified, by Jim Keogh and Ken Davidson, ISBN:0072253592, McGraw-Hill/Osborne © 2004.

1- الجزء الاول،الطبعة الاولى ،جون ر. هيوبارد، الدار الدولية سلسلة ملخصات شوم للبرمجة، بلغة C++ للاستثمارات الثقافية، ٢٠٠٠

3- هياكل البيانات / الطبعة الثانية، تاليف د.عصام الصفار،اصدارات السفير للنشر/ بغداد، ٢٠٠١

٤-الحقيبة التعليمية مادة "هياكل البيانات "، اعداد : نفارت الياس يوسف ،المعهد التقني كركوك