DATA STRUCTURES

&

ARRAYS

Basic Operations of a Collection ADT S

- insert(S, x)
- delete(S, x)
- search(S, x)
- findMin(S)
- findMax(S)
- findSuccessor(S, x)
- findPredecessor(S, x)

Collections ADTs

- Linear
- Non-linear

Linear ADTs

- Restricted Lists
  - Stack
  - Queue
    - Circular queue
    - Priority queue
- General Lists
  - Arrays
  - Linked list
  - Circular list
  - Doubly linked list

Non-linear ADTs

- Trees
  - Binary Trees and Types
  - Binary Search Trees and Variants
  - Threaded Binary Trees
  - Heaps
- Graphs
  - Undirected
  - Directed
- Hash Tables

Algorithm + Data Structure = Program Data structures

يمكن تعريف هياكل البيانات بانها: دراسة طرق الترابط بين نظرة المبرمجين للبيانات وعلاقة المعلومات بالاجهزة خصوصاً ذاكرة الحاسوب التي تخزن فيها البيانات.
هياكل البيانات تشمل طرق تنظيم المعلومات ، والخوارزميات الكفوءة في الوصول وطرق التعامل معها او تداولها (كالاضافة والحذف والتحديث والترتيب والبحث الخ) لذا فان الاهتمام لا ينحصر فقط باساليب الخزن وخوارزمياته لان الاهمية الحيوية هي قياس كلفة كل اسلوب من تلك الاساليب ومدى ملائمة استخدامها في الحالات المختلفة.

كيفية اختيار الهيكل البياني المناسب:

لكل مجموعة من البيانات هنالك اكثر من طريقة لتنظيمها ووضعها في هيكل بياني معين ويتحدد ذلك وفق عدد من العوامل والاعتبارات لاختيار الهيكل البياني المناسب وهي:

١- حجم البيانات
٢- سرعة وطريقة استخدام البيانات
٣- الطبيعة الديناميكية للبيانات كتغييرها وتعديلها دورياً
٤- السعة الخزنية المطلوبة
٥- الزمن اللازم لاسترجاع اية معلومة من الهيكل البياني
٦- اسلوب البرمجة

من العوامل المهمة في معالجة البيانات و الحصول علي النتائج المطلوبة بطرق كفؤ هو ضرورة معرفة طرق تمثيلها و اساليب التعامل معها.
لذا فان هياكل البيانات لا تعني تمثيل البيانات في هياكل معينه بل قياس متطلباتها من حيث المساحة التخزينية و الوقت.

للتعامل مع الهياكل هناك نوعان من الصيغ:

- الهيكل الفيزيائي/المادي.
  الحيز الذي تخزن فيه البيانات في الذاكرة (في صورة مصفوفة احادية).
- الهيكل المنطقي.
  الاسلوب البرمجي للتعامل مع هذه البيانات.

انواع هياكل البيانات

١- المصفوفة ARRAY

٢- القيد (السجل) RECORD

٣- الملف FILE

٤- الهياكل الخطية LINEAR STRUCTURES

- ✓ القوائم LISTS
- ✓ المكدس STACK
- ✓ الطابور QUEUE

٥- الهياكل غير الخطية NON LINEAR STRUCTURES

- ✓ المخططات GRAPHS
- ✓ الشجرTREES

المصفوفة

مجموعة من المواقع الخزنية في الذاكرة تتصف بما يلي:

- ✓ جميع المواقع من نوع بياني واحد.
- ✓ يمكن الوصول عشوائيا الي أي عنصر random Access .
- ✓ مواقع عناصر المصفوفة ثابته ولا تتغير اثناء التعامل معها.
- ✓ تمثل المصفوفة في مواقع متعاقبة بالذاكرة.

- An array as a collection of variables of the same type.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.
- A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays:

To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows:

```
type arrayName [ arraySize ];
```

This is called a single-dimension array.
The **array Size** must be an integer constant greater than zero and **type** can be any valid C++ data type.
For example, to declare a 10-element array called balance of type double, use this statement:

```
double balance[10];
```

Initializing Arrays:

You can initialize C++ array elements either one by one or using a single statement as follows:

```
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

The number of values between braces { } is the same number of elements that we declare between square brackets [ ].

Following is an example to assign a single element of the array:

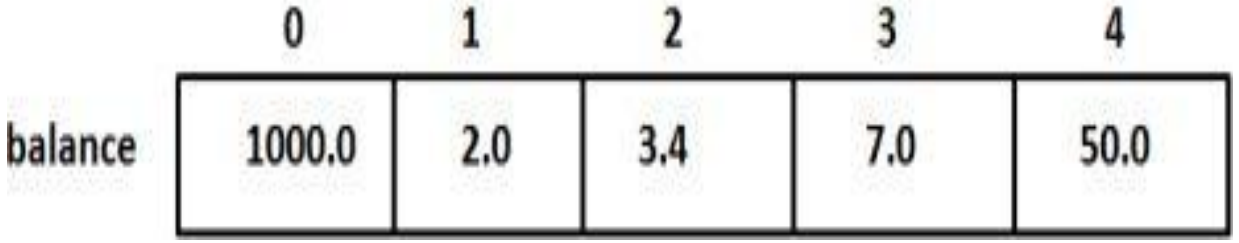
If you omit (حذف) the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write:

```
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

You will create exactly the same array as you did in the previous example.

```
balance[4] = 50.0;
```

The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th ie. last element because all arrays have 0 as the index of their first element which is also **called base index**. This array can be represented as follows:



Accessing Array Elements (الوصول):

An element is accessed **by indexing** the array name. This is done by placing the index of the element within square brackets after the name of the array. For example:

```
double salary = balance[9];
```

The above statement will take 10th element from the array and assign the value to salary variable.

Following is an example which will use all the above mentioned three concepts (declaration, assignment and accessing arrays):

```
#include <iostream>
#include <iomanip>
int main ()
{
   int n[ 10 ]; // n is an array of 10 integers
   for ( int i = 0; i < 10; i++ )
   {
      n[i]=i+100;//set element at location i to i+100
   }
   cout << "Element" << setw( 13 ) << "Value" << endl;

   // output each array element's value
   for ( int j = 0; j < 10; j++ )
   {
      cout<<setw( 7 )<< j <<setw(13)<< n[ j ] <<endl;
   }
   return 0;
}
```

This program makes use **setw()** function **to format the output**. When the above code is compiled and executed, it produces following result:

Element	Value
0	100
1	101
2	102
3	103
4	104
5	105
6	106
7	107
8	108
9	109

C++ Arrays in Detail:

Arrays are important to C++ and should need lots of more detail. There are following few **important concepts** (المفاهيم) which should be clear to a C++ programmer:

Concept	Description
Multi-dimensional arrays	C++ supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.
Pointer to an array	You can generate a pointer to the first element of an array by simply specifying the array name, without any index.
Passing arrays to functions	You can pass to the function a pointer to an array by specifying the array's name without an index.
Return array from functions	C++ allows a function to return an array.

Multi-Dimentional arrays

C++ allows multidimensional arrays. Here is the general form of a multidimensional array declaration:

```
type name[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array:

```
int threedim[5][10][4];
```

Two-Dimensional Arrays:

The simplest form of the multidimensional array is the two-dimensional array. To declare a two-dimensional integer array of size x,y you would write something as follows:

```
type arrayName [ x ][ y ];
```

Where **type** can be any valid C++ data type and **arrayName** will be a valid C++ identifier.

A two dimensional array can be think as a table which will have **x** number of **rows** and y number of columns. A 2-dimentional array **a** which contains three rows and four columns can be shown as below:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in array a is identified by an element name of the form **a[i][j]**, where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

Initializing Two-Dimensional Arrays:

Multidimensioned arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a[3][4] = {
 {0, 1, 2, 3} , /*initializers for row indexed by 0 */
 {4, 5, 6, 7} ,  /*initializers for row indexed by 1 */
 {8, 9, 10, 11}  /*initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The **following initialization is equivalent to previous example**:

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements:

An element in 2-dimensional array is accessed by using the subscripts ie. row index and column index of the array. For example:

```
int val = a[2][3];
```

The above statement will take 4th element from the 3rd row of the array.

```
#include <iostream>
int main ()
{
   // an array with 5 rows and 2 columns.
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};

   // output each array element's value
   for ( int i = 0; i < 5; i++ )
      for ( int j = 0; j < 2; j++ )
      {
         cout << "a[" << i << "][" << j << "]: ";
         cout << a[i][j]<< endl;
      }
   return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```