ARRAYS ADT AND C++ IMPLEMENTATION

NATIONAL UNIVERSITY OF TECHNOLOGY (NUTECH)

DR. SAMAN RIAZ

LECTURE # 3

DATA STRUCTURES: ARRAYS(1)

- **An ordered set (sequence) with a fixed number of elements, all of the same type**,

  where the basic operation is

  direct access to each element in the array so values can be retrieved from or stored in this element.

DATA STRUCTURES: ARRAYS (2)

Properties:

- Ordered so there is a first element, a second one, etc.
- Fixed number of elements — **fixed capacity**
- Elements must be the same type (and size);
  ∴ use arrays only for homogeneous data sets.
- Direct access: Access an element by giving its location
  - The time to access each element is the same for all elements, regardless of position – in contrast to sequential access (where to access an element, one must first access all those that precede it).

How well does C/C++ implement an array ADT?

As an ADT		In C++
ordered	⟷	indices numbered 0, 1, 2, . . ., **`CAPACITY`** - 1
fixed size	⟷	**`CAPACITY`** specifies the capacity of the array
same type elements	⟷	**`element_type`** is the type of elements
direct access	⟷	subscript operator **`[]`**

DECLARING ARRAYS IN C++

```
element_type array_name[CAPACITY];
```

where

element_type is any type

array_name is the name of the array — any valid identifier

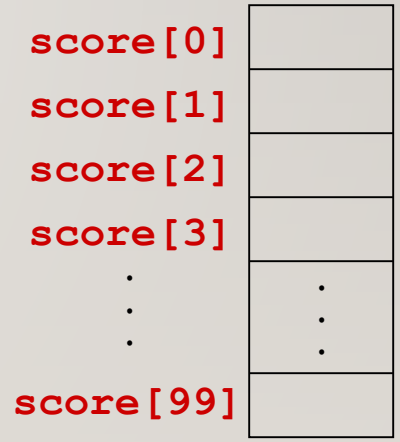
CAPACITY (a positive integer constant) is the number of elements in the array

Can't input the capacity, Why?

The compiler reserves a block of "consecutive" memory locations, enough to hold ***CAPACITY*** values of type ***element_type***.

The elements (or positions) of the array are indexed 0, 1, 2, . . ., ***CAPACITY*** - 1.

e.g., `double score[100];`



ARRAY INITIALIZATION

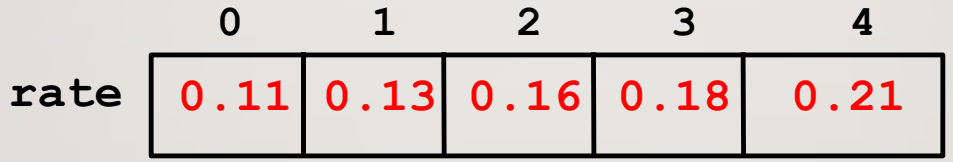
In C++, arrays can be initialized when they are declared.

Numeric arrays:

```
element_type num_array[CAPACITY] =
{list_of_initial_values};
```

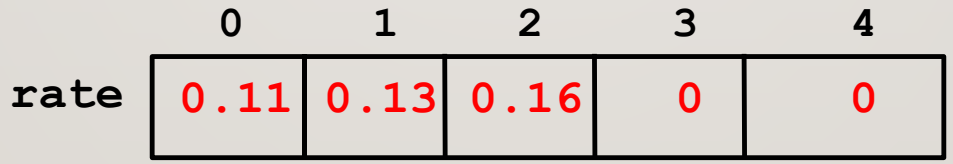
Example:

```
double rate[5] = {0.11, 0.13, 0.16, 0.18, 0.21};
```



Note 1: If fewer values supplied than array's capacity, remaining elements assigned 0.

```
double rate[5] = {0.11, 0.13, 0.16};
```



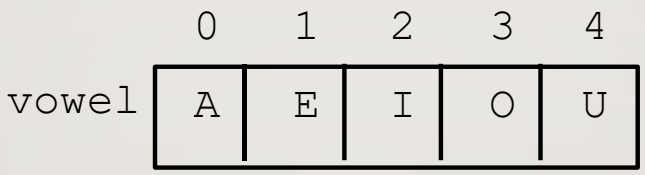
Note 2: It is an error if more values are supplied than the declared size of the array.
How this error is handled, however, will vary from one compiler to another.

Character Arrays:

Character arrays may be initialized in the same manner as numeric arrays.

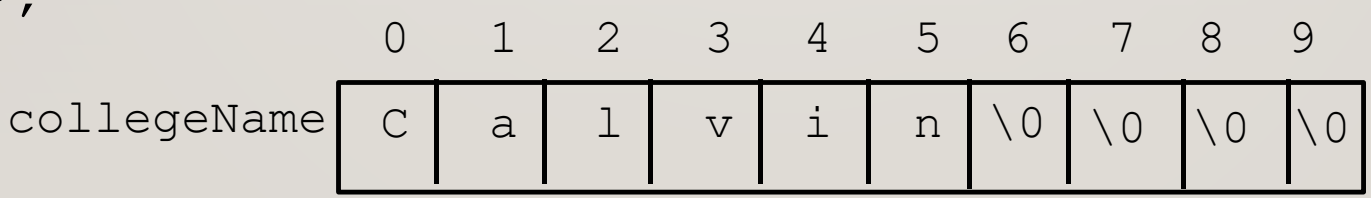
```
char vowel[5] = {'A', 'E', 'I', 'O', 'U'};
```

declares `vowel` to be an array of 5 characters and initializes it as follows:



Note 1: If fewer values are supplied than the declared size of the array, the zeroes used to fill un-initialized elements are interpreted as the null character `'\0'` whose ASCII code is 0.

```
const int NAME_LENGTH = 10;
char collegeName[NAME_LENGTH]={'C', 'a', 'l', 'v', 'i',
'n'};
```



ADDRESSES

When an array is declared, the address of the first byte (or word) in the block of memory associated with the array is called the ***base address*** of the array.

Each array reference must be translated into an ***offset*** from this base address.

For example, if each element of array **`score`** will be stored in 8 bytes and the base address of **`score`** is **`0x1396`**. A statement such as

```
cout << score[3] << endl;
```

requires that array reference **`score[3]`** be translated into a memory address:

`score[3]` → **`0x1396 + 3 * sizeof (double)`**
= 0x1396 + 3 * 8
= 0x1396 + 0x18
= 0x13ae

The contents of the memory word with this address 0x13ae can then be retrieved and displayed.

An ***address translation*** like this is carried out each time an array element is accessed.

The value of ***array_name*** is actually the ***base*** *address* **of** ***array_name***

array_name + ***index*** is the address of ***array_name[index]***.

An array reference ***array_name[index]***

is equivalent to ****(array_name + index)***

***** is the ***dereferencing*** operator

****ref*** returns the **contents of the memory location with address** ***ref***

For example, the following statements of pseudocode are equivalent:

```
print score[3]
print *(score + 3)
```

Why does an Array index start at zero (0)?

Pointers

- What is a pointer?
- How to access arrays using pointers?

PROBLEMS WITH ARRAYS

The capacity of Array can NOT change during program execution.

What is the problem?

Memory wastage

Out of range errors

C++ STYLE MULTIDIMENSIONAL ARRAYS

Most high level languages support arrays with more than one dimension.

2D arrays are useful when data has to be arranged in tabular form.

Higher dimensional arrays appropriate when several characteristics associated with data.

Example: A table of test scores for several different students on several different tests.

	Test 1	Test 2	Test 3	Test 4
Student 1	99.0	93.5	89.0	91.0
Student 2	66.0	68.0	84.5	82.0
Student 3	88.5	78.5	70.0	65.0
:	:	:	:	:
:	:	:	:	:
Student-n	100.0	99.5	100.0	99.0

For storage and processing, use a **two-dimensional array**.

Declaring Two-Dimensional Arrays

Standard form of declaration:

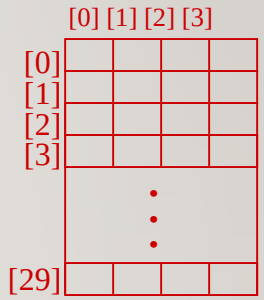
```
element_type array_name[NUM_ROWS][NUM_COLUMNS];
```

Example:

```
const int NUM_ROWS = 30,

          NUM_COLUMNS = 4;

double scoresTable[NUM_ROWS][NUM_COLUMNS];
```



Initialization

- List the initial values in braces, row by row;
- May use internal braces for each row to improve readability.

Example:

```
double rates[][3] = {{0.50, 0.55, 0.53},  // first row
                     {0.63, 0.58, 0.55}}; // second row
```

Processing Two-Dimensional Arrays

- Remember: Rows (and) columns are numbered from zero!!

- Use *doubly-indexed variables:*

  `scoresTable[2][3]` is the entry in row 3 and column 4

  row index column index

Counting from 0

- Use *nested loops* to vary the two indices, most often in a **row-wise** manner.

- Dynamic two dimensional Arrays?

Higher-Dimensional Arrays

The methods for 2D arrays extend in the obvious way to 3D arrays.

Example: To store and process a table of test scores for several different students on several different tests for several different semesters

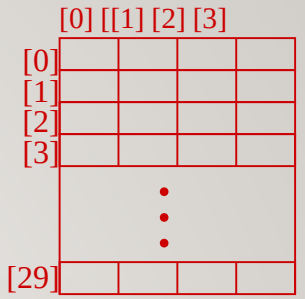
```
const int SEMS = 10, STUDENTS = 30, TESTS = 4;
typedef double ThreeDimArray[SEMS][STUDENTS][TESTS];
ThreeDimArray gradeBook;
```

`gradeBook[4][2][3]` is the score of 5th semester for student 3 on test 4

// number of semesters, students and tests all counted from zero!!

Arrays of Arrays

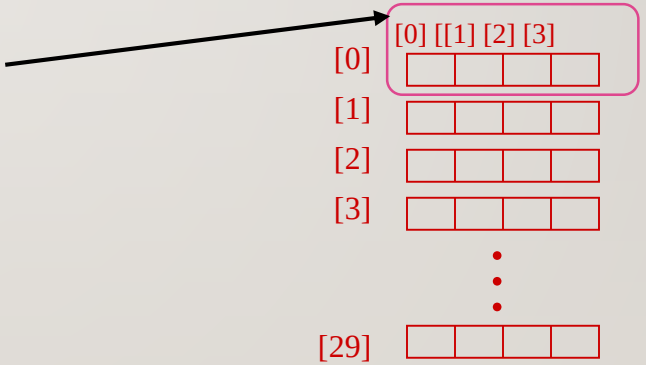
```
double scoresTable[30][4];
```



Declares **`scoresTable`** to be a one-dimensional array containing 30 elements, each of which is a one-dimensional array of 4 real numbers; that is, **`scoresTable`** is a one-dimensional array of rows , each of which has 4 real values. We could declare it as

```
typedef double RowOfTable[4];

RowOfTable scoresTable[30];
```



In any case:

`scoresTable[i]` is the **i-th row of the table**

`scoresTable[i][j]` should be thought of as **`(scoresTable[i])[j]`**
that is, as finding the j-th element of **`scoresTable[i]`**.

Address Translation:

The array-of-arrays structure of multidimensional arrays explains address translation.

Suppose the base address of **`scoresTable`** is 0x12348:

`scoresTable[10][3]`

`scoresTable[10]` → **0x12348 + 10*(sizeof RowOfTable)**

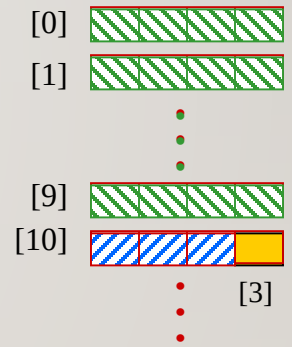
= 0x12348 + 10 * (4 * 8)

`scoresTable[10][3]`

→ **base(`scoresTable[10]`) + 3*(sizeof double)**

= 0x12348 + 10 * (4 * 8) + 3 * 8

= 0x124a0



In general, an n-dimensional array can be viewed (recursively) as a one-dimensional array whose elements are (n - 1)-dimensional arrays.

Implementing Multidimensional Arrays

- More complicated than one dimensional arrays.
- Memory is organized as a sequence of memory locations, and is thus 1D
- How to use a 1D structure to store a MD structure?

$$\begin{pmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \end{pmatrix}$$

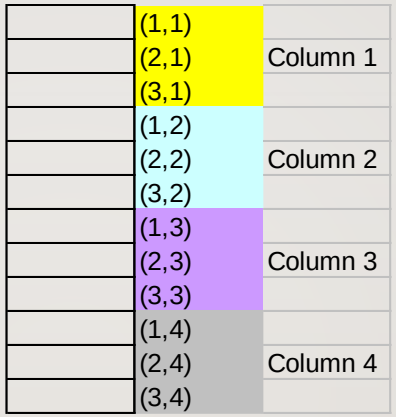
A character requires a single byte

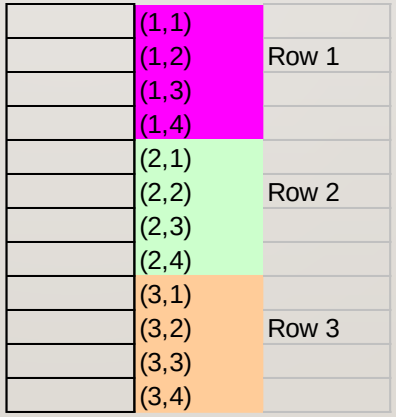
Compiler instructed to reserve 12 consecutive bytes

Two ways to store consecutively i.e. **rowwise** and **columnwise**.

TWO DIMENSIONAL ARRAYS IN MEMORY

- Two ways to be represented in memory
  - Column by column → column majored
  - Row by row → row majored





Representation depends upon the Programming language used