Data Structures

Arrays ADT and C++ Implementation

Arrays

- An array is defined as
  - Ordered collection of a fixed number of elements
  - All elements are of the same data type

- Basic operations
  - – Direct access to each element in the array –

    Values can be retrieved or stored in each element

C/C++ Implementation of an Array ADT

As an ADT	In C/C++
Ordered	Index: `0,1,2, … SIZE-1`
Fixed Size	`intExp` is constant

Homogeneous	`dataType` is the type of all elements
Direct Access	Array subscripting operator `[ ]`

Properties of an Array

- **Ordered**
  - Every element has a well-defined position
  - First element, second element, etc.
- **Fixed size or capacity**
  - Total number of elements are fixed
- **Homogeneous**
  - Elements must be of the same data type (and size)
  - Use arrays only for homogeneous data sets
- **Direct access**
  - Elements are accessed directly by their position

– Time to access each element is same

– Different to sequential access where an element is only accessed after the preceding elements

Recap: Declaring Arrays in C/C++

```
dataType arrayName[intExp];
```

- `datatype` – Any data type, e.g., integer, character, etc.
- `arrayName` – Name of array using any valid identifier
- `intExp` – Constant expression that evaluates to a positive integer

```
list[SIZE];
```

Why constant?

- Example:
  - `const int SIZE = 10;` – `int`

- Compiler reserves a block of consecutive memory locations enough to hold `SIZE` values of type `int`

Recap: Accessing Arrays in C/C++

```
arrayName[indexExp];
```

- `indexExp` – called index, is any expression that evaluates to a positive integer

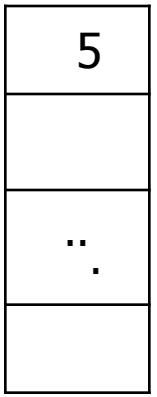
- In C/C++
  - Array index starts at 0

– Elements of array are indexed `0, 1, 2, …, SIZE-1`

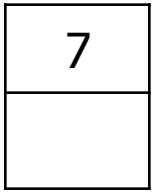
– `[ ]` is called array subscripting operator

`2;`

list[0] list[1] list[2] list[3]



list[9]



- Example

– `int value = list[2];`

– `list[0] = value +`

Array Initialization in C/C++ (1)

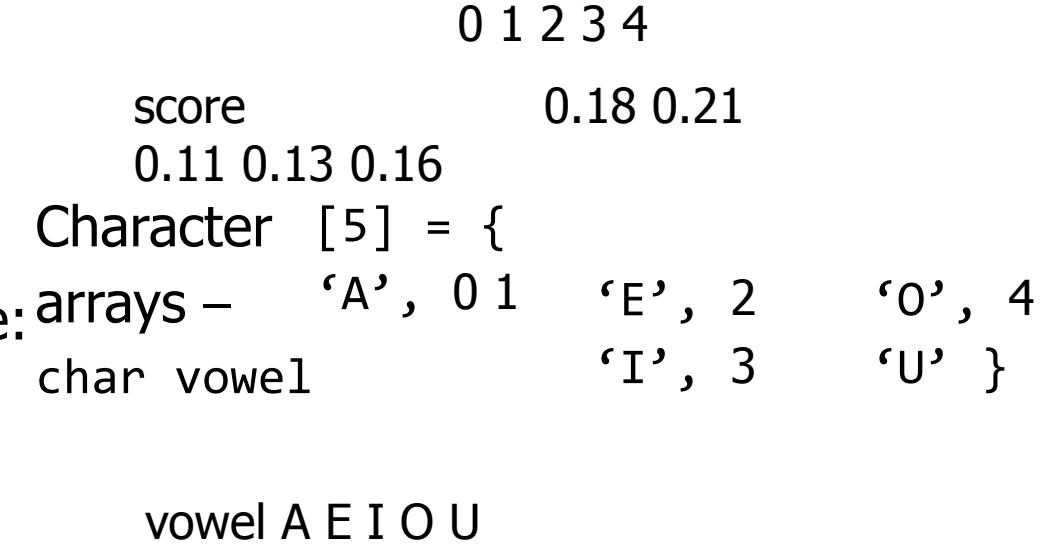
```
dataType arrayName[intExp]= {list of values}
```

- In C/C++, arrays can be initialized at declaration –

  `intExp` is optional: Not necessary to specify the size

- Example: Numeric arrays

- double score[ ] = {0.11, 0.13, 0.16, 0.18, 0.21}

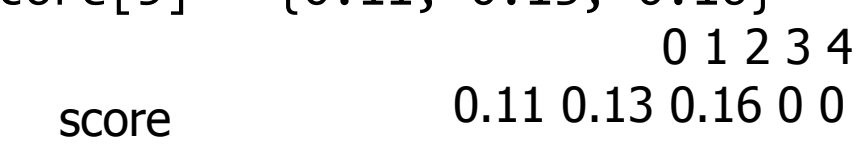


- Example:

Array Initialization in C/C++ (2)

- Fewer values are specified than the declared size of an array –
  Numeric arrays: Remaining elements are assigned zero – Character arrays: Remaining elements contains null character '\0' ➢ ASCII code of '\0' is zero
- Example

- `double score[5] = {0.11, 0.13, 0.16}`



- `char name[6] = {'J', 'O', 'H', 'N'}`

- If more values are specified than declared size of an array
  - Error is occurred: Handling depends on compiler

Multidimensional Arrays

- Most languages support arrays with more than one dimension – High dimensions capture characteristics/correlations associated with data

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

– 2D array is suitable for storage and processing of data

	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

Two Dimensional Arrays – Declaration

```
dataType arrayName[intExp1][intExp2];
```

- `intExp1` – constant expression specifying number of rows •

`intExp2` – constant expression specifying number of columns

- Example:
  - `const int NUM_ROW = 2, NUM_COLUMN = 4;`
  - `double scoreTable [NUM_ROW][NUM_COLUMN];`

- Initialization:
  -
```
Double scoreTable [ ][4] = { {0.5, 0.6, 0.3},
                             {0.6, 0.3,
                             0.8}};
```
  - List the initial values in braces, row by row
  - May use internal braces for each row to improve readability

Two Dimensional Arrays – Processing

```
arrayName[indexExp1][indexExp2];
```

- `indexExp1` – row index
- `indexExp2` – column index

- Rows and columns are numbered from 0
- Use nested loops to vary two indices
  - Row-wise or column-wise manner

sco [0] [1] [2] [3] re

[0] [1][2][3]

	0.7		
..	..	..	..

- Example

– `double value = score[2][1]; – score[0][3] = value + 2.0;`

[9]

			2.7

Higher Dimensional Arrays

- Example: Store and process a table of test scores
  - For several different students
  - On several different tests
  - Belonging to different semesters

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

- What is represented by `gradebook[4][2][3]`?
  - Score of 3rd student belonging to 5th semester on 4th test

- All indices start from zero