AAB University

Faculty of Computer Sciences

Object Oriented Programming

Week 7:

Class Templates Array and Vector

Asst. Prof. Dr. **Mentor Hamiti**

mentor.hamiti@universitetiaab.com

Last Time?!

- **Standard Library Functions**
  - Math Library Functions
- **User Defined Functions**
  - Standard Functions
  - Inline Functions
  - Macro Functions
- **More on Functions**
  - References and Reference Parameters
  - Defaults Arguments
  - Unary Scope Resolution Operator
  - Function Overloading
- **Recursions**

Today

- Arrays and Vectors
- Examples Using Arrays
- Standard Library Class Template: **array** and **vector**

Arrays and Vectors

- A **one-dimensional Array** (Vector) is a structured composite data type made up of a finite, fixed-size collection of ordered homogeneous elements to which there is direct access
  - **A finite**
  - **Fixed-size**
  - **Ordered**
  - **Homogeneous**
  - **Direct access**

What operations are defined for the array?

- Example:

```
int numbers [10]
```

numbers

[0]	Firs element
[1]	Second element
[2]	Third element
.	.
.	.
.	.
[9]	Last element

Arrays

- An array declaration statements tells the compiler how many cells are needed to represent that array
- The name of the array then is associated with the characteristics of the array
- These characteristics include:

1. The number of elements (***Number***)
2. The location in memory of the first cell in the array, called the base address of the array (***Base***)
3. The number of memory locations needed for each element in the array (***SizeOfElement***)

Arrays

- Example:

  int *data* [10]
  float *money* [6]
  char *letters* [26]

- *In C++, the accessing function that gives us the position of an element in a one-dimensional array associate with the expression Index is:*

  Addres(Index) = Base + Index * SizeOfElement

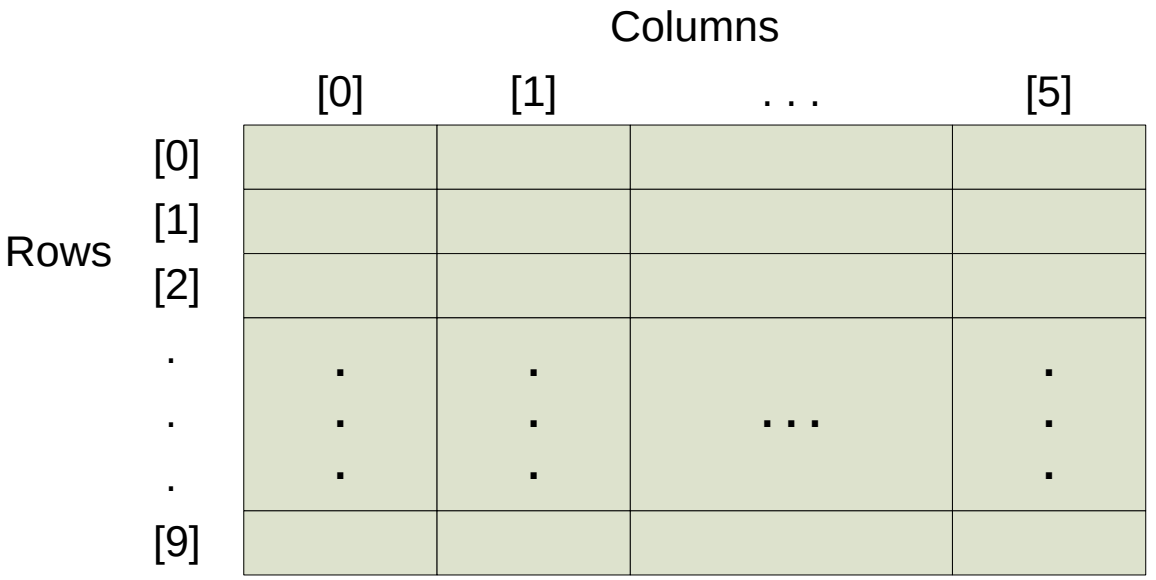
data [0]	100 + (0 * 4)	Address 100
data [8]	100 + (8 * 4)	Address 132
letters [1]	160 + (1 * 1)	Address 161
money [3]	140 + (3 * 4)	Address 152
letters [25]	160 + (25 * 1)	Address 185

Arrays

- A two-dimensional array is a natural way to represent data that is logically viewed as a table with a columns and rows:

```
const int row;
const int col;
int table [row][col]
```

int *table* [10][6]



Arrays

- **Index** is a finite ordered set of one or more dimensions, for example,
  - For one dimension:

    [n]

    {0, … , n-1}

    Running time?

  - For two dimensions:

    [3][3]

    {(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)}

  - For three dimensions:

    [2] [2] [2]

    {(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1)}

- Arrays and Vectors
- Examples Using Arrays

Declaring an Array and Using a Loop for Initialization

- Example 1:

```
//Initializing an array's elements to zeros and printing the array
# include <iostream>
# include <iomanip>
using namespace std;

int main()
{
    int n[ 10 ];    // n is an array of 10 integers

    // initialize elements of array n to 0
    for ( int i = 0; i < 10; ++i )
        n[ i ] = 0; // set element at location i to 0

    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;

    cin.get();
    return 0;
}
```

```
Element        Value
      0            0
      1            0
      2            0
      3            0
      4            0
      5            0
      6            0
      7            0
      8            0
      9            0
```


©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Initializing an array in a Declaration

- Example 2:

```
// Initializing an array in a declaration.
# include <iostream>
# include <iomanip>
using namespace std;

int main()
{
    // use initializer list to initialize array n
    int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

    cout << "Element" << setw( 13 ) << "Value" << endl;

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

cin.get();
return 0;
}
```

```
Element         Value
      0            32
      1            27
      2            64
      3            18
      4            95
      5            14
      6            90
      7            70
      8            60
      9            37
```


©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Specifying an array's Size with a Constant Variable

- Example 3:

```
// Set array s to the even integers from 2 to 20
# include <iostream>
# include <iomanip>
using namespace std;

int main()
{

    const int arraySize = 10;

    int s[ arraySize ]; // array s has 10 elements

    for ( int i = 0; i < arraySize; ++i ) // set the values
        s[ i ] = 2 + 2 * i;

    cout << "Element" << setw( 13 ) << "Value" << endl;

    // output contents of array s in tabular format
    for ( int j = 0; j < arraySize; ++j )
        cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;

cin.get();
return 0;
}
```

```
Element         Value
      0             2
      1             4
      2             6
      3             8
      4            10
      5            12
      6            14
      7            16
      8            18
      9            20
```


©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Elements of an array used in calculation

- Example 4:

```
// Compute the sum of the elements of the array
# include <iostream>
using namespace std;

int main()
{
    const int arraySize = 10; // constant variable indicating size of array
    int a[ arraySize ] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };

    int total = 0;

    for ( int i = 0; i < arraySize; ++i )
        total += a[ i ];

    cout << "Total of array elements: " << total << endl;

cin.get();
return 0;
}
```

```
Total of array elements: 849
```


©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Using Bar Charts to Display array Data Graphically

- Many programs present data to users in a graphical manner
- One simple way to display numeric data graphically is with a **bar chart** that shows each numeric value as a bar of asterisks (*)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Using Bar Charts to Display array Data Graphically

- Example 5:

```
// Bar chart printing program
# include <iostream>
# include <iomanip>
using namespace std;

int main()
{
    const int arraySize = 11;
    int n[ arraySize ] = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };

    cout << "Grade distribution:" << endl;

    // for each element of array n, output a bar of the chart
    for ( int i = 0; i < arraySize; ++i )
    {
    // output bar labels ("0-9:", ..., "90-99:", "100:" )
    if ( i == 0 )
        cout << " 0-9: ";
    else if ( i == 10 )
        cout << "  100: ";
    else
        cout << i * 10 << "-" << ( i * 10 ) + 9 << ": ";
    // print bar of asterisks
    for ( int stars = 0; stars < n[ i ]; ++stars )
        cout << '*';
    cout << endl; // start a new line of output
    } // end for

    cin.get(); return 0;
}
```

```
Grade distribution:
 0-9:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```


©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Using the Elements of an array as Counters

- Many programs present data to users in a graphical manner
- One simple way to display numeric data graphically is with a **bar chart** that shows each numeric value as a bar of asterisks (*)

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Using the Elements of an array as Counters

- Example 6:

```
Face      Frequency
   1        1000319
   2        1001198
   3         999219
   4         998883
   5        1000363
   6        1000018
```

```
// Roll a six-sided die 6,000,000 times
# include <iostream>
# include <iomanip>
# include <cstdlib>
# include <ctime>
using namespace std;

int main()
{
    const int arraySize = 7;            // ignore element zero
    int frequency[ arraySize ] = {};    // initialize elements to 0

    srand( time( 0 ) );                 // seed random number generator

    // roll die 6,000,000 times; use die value as frequency index
    for ( int roll = 1; roll <= 6000000; ++roll )
        ++frequency[ 1 + rand() % 6 ];

    cout << "Face" << setw( 13 ) << "Frequency" << endl;

    // output each array element's value
    for ( int face = 1; face < arraySize; ++face )
        cout << setw( 4 ) << face << setw( 13 ) << frequency[ face ]<< endl;

cin.get(); return 0;
}
```


©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Using arrays to Summarize Survey Results

- Example 7:

```
// Poll analysis program
# include <iostream>
# include <iomanip>
using namespace std;

int main()
{
    // define array sizes
    const int responseSize = 20; // size of array responses
    const int frequencySize = 6; // size of array frequency

    // place survey responses in array responses
    const int responses[ responseSize ] = { 1, 2, 5, 4, 3, 5, 2, 1, 3,
                                            1, 4, 3, 3, 3, 2, 3, 3, 2, 2, 5 };

    // initialize frequency counters to 0
    int frequency[ frequencySize ] = {};

    // for each answer, select responses element and use that value
    // as frequency subscript to determine element to increment
    for ( int answer = 0; answer < responseSize; ++answer )
        ++frequency[ responses[ answer ] ];

    cout << "Rating" << setw( 17 ) << "Frequency" << endl;

    // output each array element's value
    for ( int rating = 1; rating < frequencySize; ++rating )
        cout << setw( 6 ) << rating << setw( 17 ) << frequency[ rating ]<< endl;

    cin.get(); return 0;
}
```

```
Rating          Frequency
     1                  3
     2                  5
     3                  7
     4                  2
     5                  3
```


©1992-2014 by Pearson Education, Inc. All Rights Reserved.

- Arrays and Vectors
- Examples Using Arrays
- Standard Library Class Template: **array** and **vector**

Arrays

- An **array** is a contiguous group of memory locations that all have the same type
- To specify the type of the elements and the number of elements required by an array use a declaration of the form:

  array< type, arraySize > *arrayName*;

  - The notation **<type, arraySize>** indicates that array is a **class template**
  - The compiler reserves the appropriate amount of memory based on the type of the elements and the arraySize arrays can be declared to contain values of most data types

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

*Range-Based **for** Statement*

- It's common to process all the elements of an array
- The new C++ **range-based for statement** allows the possibility to do this without using a counter, thus avoiding the possibility of "stepping outside" the array and eliminating the need for bounds checking
- The syntax of a range-based for statement is:

```
for ( rangeVariableDeclaration : expression )
    statement
```

  - rangeVariableDeclaration *has a type and an identifier* (e.g., int item), *and expression is the array through which to iterate*
  - *The type in the* rangeVariableDeclaration *must be consistent with the type of the array's elements*

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

C++ Standard Library Class Template **array**

- Example 8:

```
//Range-Based for Statement
# include <iostream>
# include <array>
using namespace std;

int main()
{
    array <int, 5> items = { 1, 2, 3, 4, 5};

    cout<<"\nItems before modificaton: ";
        for(int item: items)
        cout<<item<<"  ";

    //multiply the elements of items by 2
    for(int &itemR: items)
        itemR *= 2;

    cout<<"\n\nItems after  modificaton: ";
    for(int item: items)
        cout<<item<<"  ";

        cin.get(); return 0;
}
```

```
Items before modificaton: 1  2  3  4  5

Items after  modificaton: 2  4  6  8  10
```


©1992-2014 by Pearson Education, Inc. All Rights Reserved.

Language built-in array vs Library array

```
#include <iostream>

using namespace std;

int main()
{
  int myarray[3] = {10,20,30};

  for (int i=0; i<3; ++i)
    ++myarray[i];

  for (int elem : myarray)
    cout << elem << '\n';

  cin.get(); return 0;
}
```

```
11
21
31
```

```
#include <iostream>
#include <array>
using namespace std;

int main()
{
    array<int,3> myarray = {10,20,30};

    for (int i=0; i<myarray.size(); ++i)
        ++myarray[i];

    for (int elem : myarray)
        cout << elem << '\n';

  cin.get(); return 0;
}
```

```
11
21
31
```

- *Both kinds of arrays use the same syntax to access its elements: myarray[i]. Other than that, the main differences lay on the declaration of the array, and the inclusion of an additional header for the library array. Notice also how it is easy to access the size of the library array*

*C++ Standard Library Class Template **vector***

- The Standard Template Library's **vector**s are somewhat like **array**s but are considerably more flexible
  - *Unlike a typical **array**, a **vector** can grow and shrink in size as needed, although there can sometimes be a performance penalty when the size is increased*
- By default, all the elements of a vector object are set to **0**

```
vector <int> myVector (10);
```

- Vector member function **size** obtain the number of elements in the vector

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

- Example 9:

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector <int> myVector;          //Vector to store integers
    myVector.push_back(3);          //Add 3 onto the vector
    myVector.push_back(10);         //Add 10 to the end
    myVector.push_back(33);         //Add 33 to the end

    for(int x=0; x<myVector.size(); x++)
    {
        cout<<myVector[x]<<" ";     //Output: 3 10 33
    }

  cin.get(); return 0;
}
```

```
3 10 33
```


©1992-2014 by Pearson Education, Inc. All Rights Reserved.

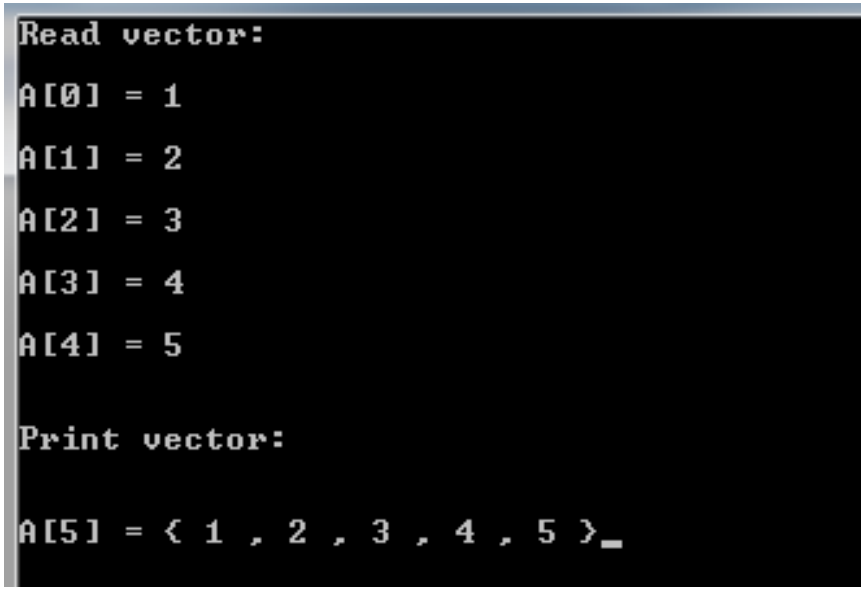
C++ SL Class Template: ***array*** *and* ***vector***

- One of the key differences between a **vector** and an **array** is that a vector can dynamically grow to accommodate more elements

©1992-2014 by Pearson Education, Inc. All Rights Reserved.

- Exercise 1:

Write the C++ code for having the following output:



- Exercise 2:

  Write the C++ code for having the following output:

```
Read matric:
A[0][0] = 1
A[0][1] = 2
A[0][2] = 3
A[0][3] = 4
A[1][0] = 5
A[1][1] = 5
A[1][2] = 5
A[1][3] = 5
A[2][0] = 1
A[2][1] = 2
A[2][2] = 3
A[2][3] = 4

Print matric:

        1       2       3       4
        5       5       5       5
        1       2       3       4
```

- Project 1 (*for bonus points* ☺):

```
//Complete the following code for sorting matrice:
# include .....
.....................
int main()
{
        readMatrice(A,n,m);
        printMatrice(A,n,m);
        convertMatriceToVector(A,n,m,V);
        writeVector(V);
        sortVector(V);
        writeVector(V);
        convertVectorToMatrice(V,n,m,A);
        printMatrice(A,n,m);
cin.get();    return 0;
}
```

- Questions?!

mentor.hamiti@universitetiaab.com