**Multiple-Dimension Array, ArrayList, and List<T> Class**

**Introduction**

In one of the previous lectures, you had learned to create single-dimension array in C++ and managed C++. This lecture will discuss the concept of jagged array and methods to create multiple dimension arrays in managed code. Arrays must have a fixed size, yet it often causes issues when there is a need to continuously enlarge the array size. This lecture will then introduce the .NET Framework’s ArrayList and List<T> classes. They both support dynamic size and are not limited to a fixed size. The number of their elements can continuously increase.

**Managed array**

For the sake of review, this section discusses how to create a managed single dimension array. The System::Array class provides methods for creating, manipulating, searching, and sorting arrays, thereby serving as the base class for all arrays in the common language runtime. The syntax to declare a managed array is:

```csharp
array<type^> ^arrayName = gcnew array<type^>(size);
```

where,

- `type` is the data type, such as String, int, double, float, and so on.
- `arrayName` is the name of the array.
- `size` is the length of an Array is the total number of elements it can contain.

For example, to declare a double type of managed array (named `a`) with 5 elements:

```csharp
array<double ^> ^a = gcnew array<double ^>(5);
```

Since an array has elements, and each element has an order, learning to declare an array in Visual C++ is based on how to define the list with correct sequence of each element in the array.

The seasons have string-based values, so you can declare a string array similar to:

```csharp
array<String ^> ^season = gcnew array<String ^>(4);
```

**Creating elements and assigning their values**

Once the array is declared, you need to create elements and assign valid values to them. There are two different ways, formal and short-hand ways.

The formal way requires you to clearly indicates the index (also known as key) value of the element and associate it with a valid value. The syntax is:

```csharp
arrayName[index] = "value";
```

where `index` are integers starting with 0. The first element is indexed as 0, the second 1, the third 2, and so on. If value are numbers, double quotes (""""") are omitted. For example,

```csharp
array<double ^> ^a = gcnew array<double ^>(5);
a[0] = 1.23;
a[1] = 0.44;
a[2] = 3.26;
a[3] = 3.57;
a[4] = 6.74;
```

The shorthand way is a simplified way. With which, element values are directly assigned in an initializer list, as shown below:

```csharp
array<double ^> ^a = gcnew array<double ^>(5){1.23, 0.44, 3.26, 3.57, 6.74};
```
In the season array, you can use

```csharp
array<String ^> ^ season = gcnew array<String ^>(4);
season[0] = "String";
season[1] = "Summer";
season[2] = "Fall";
season[3] = "Winter";
```

or simply:

```csharp
array<String ^> ^ season = gcnew array<String ^>(4)
{"String","Summer","Fall","Winter"};
```

To retrieve the value held by each element in an array, use the following syntax:

```csharp
arrayName[index]
```

For example,

```csharp
MessageBox::Show(a[3]);
```

A simple repetition structure can help retrieving each element’s value. For example,

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main() {
    String^ str="";
    array<String ^> ^ season = gcnew array<String ^>(4);
season[0] = "String";
season[1] = "Summer";
season[2] = "Fall";
season[3] = "Winter";

for (int i=0; i < season->Length; i++) {
    str += season[i] + "\n";
}

MessageBox::Show(str);
return 0;
}
```

The `Length` property gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.

The `for each` statement is frequently used to iterate through the element of an entire array. The syntax is:

```csharp
for each (type identifier in arrayName) { ..... }
```

where `type` and `identifier` are the type and name of the iteration variable. For example,

```csharp
for each (String^ I in season) {
    str += I + "\n";
}
MessageBox::Show(str);
```
A **jagged array** is an array whose elements are arrays. A jagged array is an “array of arrays”. The instructor creates the following sample code to depict the concept of “array of arrays”. The code starts with creates three int type of arrays: x, y, and z. Their sizes are 4, 2, and 3 respectively. The code then creates another array named “a” which uses x, y, and z as elements.

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    array<int>^ x = gcnew array<int> {1, 2, 3, 4};
    array<int>^ y = gcnew array<int> {5, 6};
    array<int>^ z = gcnew array<int> {7, 8, 9};

    array<array<int>^>^ a = gcnew array<array<int> ^>(3) { x, y, z};
}
```

The above code constitute the logic of jagged array; however, it is not the suggested way to create a jagged array because it reverses the order of steps. The following declares a jagged array which is an array of **int** arrays. You can consider this array as a “parent” array which has several “child” arrays. Interestingly, the elements of a jagged array can be of different dimensions and sizes.

```csharp
array<array<int>^>^ a;
```

The following initializes the array and specifies that it must have three **int** arrays as elements.

```csharp
a = gcnew array<array<int> ^>(3);
```

You can combine the above statements in one lines:

```csharp
array<array<int>^>^ a = gcnew array<array<int> ^>(3);
```

The first element of the “a” array is a[0] which is an array by itself. The second element is a[1] and the third is a[2]. The following is a sample way to declare and initialize a[0] as an array of four **int** elements.

```csharp
a[0] = gcnew array<int>(4);
```

Every element of the a[0] array is identified as a[0][j], where j is the index of elements in a[0]. The following assigns 1, 2, 3, and 4 are values to a[0][0], a[0][1], a[0][2], and a[0][3].

```csharp
a[0][0] = 1;
a[0][1] = 2;
a[0][2] = 3;
a[0][3] = 4;
```

A short-hand way to declare and initialize a[0] is:

```csharp
a[0] = gcnew array<int>(4) {1, 2, 3, 4};
```

The following examples show how to declare, initialize, and access jagged arrays. It uses the format a[i][j] to identify values of the jagged array, where i is index of the parent and j is index of the child arrays.

```csharp
#using <System.dll>
```
The above code uses two for loop to retrieve values recursively. The size of array “a” is represented by a->Length. Since every element in “a” varies in size, the instructor uses a[i]->Length to dynamic retrieve the size of every element in “a”.

The following is another example of jagged array. Notice that all its elements are initialized through the short-hand way.

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    array<array<String^>^>^ food = gcnew array<array<String^> ^>(3);  
        {  
            {"apple", "banaba", "orange"}, // 1st element
            {"onion", "pepper"}, //2nd element
            {"broccoli", "potato", "tomato", "spinach"} //3rd element
        };
    
    String^ str;
    for (int i=0; i < food->Length; i++)
    {
        for (int j=0; j < food[i]->Length; j++)
        {
            str += food[i][j] + "\n";
        }
    }
    MessageBox::Show(str);
}
```
for (int j=0; j < food[i]->Length; j++)
{
    str += food[i][j] + "\n";
}
}
MessageBox::Show(str);

Multi-dimensional arrays

A two dimensional array is an array that takes two indexes to reference an element. It is built on top of a two dimensional table, as such below, which is a table with 2 rows and 3 columns.

<table>
<thead>
<tr>
<th>Col0</th>
<th>Col1</th>
<th>Col2</th>
<th>Col3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[0, 0]</td>
<td>A[0, 1]</td>
<td>A[0, 2]</td>
<td>A[0, 3]</td>
</tr>
</tbody>
</table>

In this array A, the value of the intersection, row1 and col3, is represented by A[0, 2]. To declare such array, use the following syntax:

```cpp
array< type, rank>^ arrayName = gcnew array< type, rank>(size);
```

where the rank of an Array is the number of dimensions in the Array. For example, the A array can be declared use:

```cpp
array<int, 2>^ A = gcnew array<int, 2>(2,4);
```

To create elements and assign values to them, use:

A[0, 0] = 53;
A[0, 1] = 97;
A[0, 2] = 46;
A[0, 3] = 81;
A[1, 0] = 77;
A[1, 1] = 64;
A[1, 2] = 82;
A[1, 3] = 68;

By the way, the Rank property to get the rank (number of dimensions) of the Array. For example, the following code will return 2.

```cpp
array<int, 2>^ A = gcnew array<int, 2>(2,4);
MessageBox::Show(A->Rank + "]");
```

The following table illustrates how the values are assigned.

<table>
<thead>
<tr>
<th>Col0</th>
<th>Col1</th>
<th>Col2</th>
<th>Col3</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>97</td>
<td>46</td>
<td>81</td>
</tr>
<tr>
<td>77</td>
<td>64</td>
<td>82</td>
<td>68</td>
</tr>
</tbody>
</table>

A short-hand way is to consider the two-dimensional array a nested array. Let each row alone be an array, so you can use:

```cpp
array<int, 2>^ A = gcnew array<int, 2>(2,4)
{(53,97,46,81),(77,64,82,68)};
```

To retrieve the elements, use two for loops. One handles rows; the other handles columns.

```cpp
#using <System.dll>
```
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main() {
    String^ str="";

    array<int, 2>^ A = gcnew array<int, 2>(2,4) {{53,97,46,81},{77,64,82,68}};

    for (int i=0; i < 2; i++) { // row
        for (int j=0; j < 4; j++) { // column
            str += A[i,j] + "\n";
        }
    }
    MessageBox::Show(str);
    return 0;
}

To automatically retrieve the size (number of element) of array on a per-row-per-column basis, use the Array::GetLength(n) method gets a 32-bit integer that represents the number of elements in the specified dimension (n) of the Array. The 0th dimension consists of rows, the 1st dimension consists of columns.

#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main() {
    String^ str="";

    array<int, 2>^ A = gcnew array<int, 2>(2,4) {{53,97,46,81},{77,64,82,68}};

    for (int i=0; i < A->GetLength(0); i++) { // row
        for (int j=0; j < A->GetLength(1); j++) { // column
            str += A[i,j] + "\n";
        }
    }
    MessageBox::Show(str);
    return 0;
}

To declare a two-dimensional native array, use the following syntax:

type arrayName[rowSize][columnSize];

For example,

    int a[2][4] = {{53,97,46,81},{77,64,82,68}};

which is the short-hand form of the following:

    int a[2][4];
    a[0][0] = 53;
    a[0][1] = 97;
    a[0][2] = 46;
a[0][3] = 81;
a[1][0] = 77;
a[1][1] = 64;
a[1][2] = 82;
a[1][3] = 68;

To display all the elements, use two for loops as shown below.

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;

int main() {
    String^ str="";
    int a[2][4] = {{53,97,46,81},{77,64,82,68}};
    for (int i=0; i < 2; i++) {
        for (int j=0; j < 4; j++) {
            str += a[i][j] + "\n";
        }
    }
    MessageBox::Show(str);
    return 0;
}
```

The length of a two-dimensional native array can be calculated by using the sizeof operator. Again, when the sizeof operator is applied to an array, it yields the total number of bytes in that array, not the size of the pointer represented by the array identifier. The sizeof operator can return the actual size of its operand in byte. The following illustrates its syntax where operand could be an object or a variable.

```
sizeof(operand)
```

The following is sample code that demonstrates how to use the sizeof operator to determine the physical size of the variable `x` and `str`.

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;

int main()
{
    int x = 5;
    String^ str = "Hello world!";
    MessageBox::Show(sizeof(x) + "\n" + sizeof(str));
}
```

The following is a sample output of the above code which indicates both int and String are 4 bytes in size.

```
4
4
```

The sizeof operator is a C++ tool created for checking the physical size in memory of a given class or a variable of a specific C++ data type. It is used when actual size of the object must be
known. It can also be used to determine the physical size of an array. In the following example, there are five elements in the “a” array. Visual C++ will store each of the element in an individual memory location, each memory location is 4 bytes in size. However, the `sizeof` operator, when the operand is the array, will return the sum of physical memories of all its elements. The output of the following code is 20 (bytes) because 5 elements × 4 bytes = 20 bytes.

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    int a[] = {4, 5, 6, 7, 8};
    MessageBox::Show(sizeof(a) + "");
}
```

Each element in an array should have exactly the same physical size in memory. If the size of the array is 20 bytes, and each element takes 4 bytes, then there should be 20 / 4 = 5 elements. The following can provide an approach to find out the size (total number of elements) of an array.

\[
\text{sizeofArray} = \frac{\text{sizeof(arrayName)}}{\text{sizeof(dataType)}}
\]

You can create an integer array with the number of elements equals to the rank (number of dimension). In the following example, length is an integer array of two elements.

```csharp
int length[2];
```

To let the first element represents the number of rows in the two-dimensional array “a”, you need to divide the total number of the two-dimensional array “a” by the number of element each row has. The following is the implementation:

```csharp
length[0] = (sizeof(a)/sizeof(int)) / (sizeof(a[0])/sizeof(int));
```

To calculate the number of elements each row (such as the first row) has, use:

```csharp
length[1] = (sizeof(a[0])/sizeof(int));
```

With the above measure, you can re-write the above code to:

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    Form^ form1 = gcnew Form;
    form1->Width=200;
    form1->Height=200;

    Label^ label1 = gcnew Label;
    label1->AutoSize = true;
    form1->Controls->Add(label1);

    //code
    String^ str="";
```
// native array
int a[2][4] = {{53, 97, 46, 81}, {77, 64, 82, 68}};

int length[2];
length[0] = (sizeof(a)/sizeof(int)) / (sizeof(a[0])/sizeof(int));
length[1] = (sizeof(a[0])/sizeof(int));

for (int i=0; i < length[0]; i++) {
    for (int j=0; j < length[1]; j++) {
        str += a[i][j] + "\n";
    }
}

// display the content
label1->Text = str;
Application::Run(form1);

The following figure explains how a three-dimensional array works.

The following figure explains the identifiers of each element.

Consider the following code, which is a three dimensional array.

array<String^, 3> ^ clr = gcnew array<String^, 3>(2, 4, 2);
clr[0, 0, 0] = "red";
clr[0, 0, 1] = "green";
clr[0, 1, 0] = "blue";
clr[0, 1, 1] = "orange";
clr[0, 2, 0] = "black";
clr[0, 2, 1] = "gold";
clr[0, 3, 0] = "beige";
clr[0, 3, 1] = "silver";
clr[1, 0, 0] = "yellow";
clr[1, 0, 1] = "pink";
clr[1, 1, 0] = "teal";
clr[1, 1, 1] = "white";
clr[1, 2, 0] = "gray";
clr[1, 2, 1] = "cyan";
clr[1, 3, 0] = "brown";
clr[1, 3, 1] = "lime";

String^ str;

for (int i=0; i<=1; i++) {
    for (int j=0; j<=3; j++) {
        for (int k=0; k<=1; k++) {
            str += "clr[" + i + ", " + j + ", " + k + "] is " + clr[i, j, k] + 
            "\n";
        }
    }
}

MessageBox::Show(str);

The elements and their values are illustrated below:

The following is another sample code

```c++
#include <System.dll>
#include <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    array<int, 3>^ a = gcnew array<int, 3>(4,4,3);

    String^ str;
    int n=0;

    for (int i=0; i<4; i++)
    {
        for (int j=0; j<4; j++)
        {
            for (int k=0; k<3; k++)
            {
                a[i,j,k] = n+1; // assign value to each element
                str += a[i,j,k] + " ",
                n++;
            }
        }
    }

    MessageBox::Show(str);
}
```
A sample output looks:

![Image](image.png)

The **Rank** property also applies to three-dimensional arrays. The following will return 3.

```csharp
array<String^, 3> ^ clr = gcnew array<String^, 3>(2, 4, 2);
MessageBox::Show(clr->Rank + "");
```

The **GetLength()** method can be used to get the size of each dimension. The 0th dimension consists of rows, the 1st dimension consists of columns, and the 2nd dimension consists of layers. For example,

```csharp
for (int i=0; i < clr->GetLength(0); i++) {
  for (int j=0; j < clr->GetLength(1); j++) {
    for (int k=0; k < clr->GetLength(2); k++) {
      str += "clr["+ i +", "+ j +", "+ k + "] is " + clr[i, j, k] + 
    }
  }
}
```

To create a three-dimensional native array, use:

```csharp
// native array
int n[2][4][2] = {
  {{21, 37}, {16, 9}, {54, 83}, {17, 60}},
  {{44, 98}, {17, 56}, {29, 67}, {47, 18}},
};
int length[3];
length[2] = (sizeof(n[0][0])/sizeof(int));
length[1] = (sizeof(n[0])/sizeof(int)) / length[2];
length[0] = (sizeof(n)/sizeof(int)) / (sizeof(n[0])/sizeof(int));
String^ str2 = "Native array" + Environment::NewLine;
for (int i=0; i < length[0]; i++) {
  for (int j=0; j < length[1]; j++) {
    for (int k=0; k < length[2]; k++) {
      str2 += n["+ i +"]["+ j +"]["+ k + "] is " + n[i][j][k] + Environment::NewLine;
    }
  }
}
```

You can create an integer array with three elements to handle the length of row, column, and layer. The length of row, again, is calculated by:

```csharp
int length[3];
length[0] = (sizeof(n)/sizeof(int)) / (sizeof(n[0])/sizeof(int));
```

The column is:

```csharp
length[1] = (sizeof(n[0])/sizeof(int)) / length[2];
```
The layer is:

\[
\text{length}[2] = (\text{sizeof}(n[0][0]) / \text{sizeof}(\text{int}));
\]

ListBox frequently use array to handle its elements. The following code example demonstrates the `AddRange` method and various other methods of the ListBox class that act on ranges. An array of strings is created and passed to the constructor, populating the list with the elements of the array. The AddRange method is called, with the list as its argument. The result is that the current elements of the list are added to the end of the list, duplicating all the elements.

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>
#using <System.Drawing.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;

public ref class Form1: public Form {

public:
    Form1() {
        ListBox^ listBox1 = gcnew ListBox;

        array<String ^> ^ states = gcnew array<String ^>{{
            "AK", "AL", "AR", "AZ", "CA", "CO", "CT", "DC", "DE", "FL",
            "GA", "HI", "IA", "ID", "IL", "IN", "KS", "KT", "LA", "MA",
            "MD", "ME", "MI", "MN", "MO", "MS", "MT", "NC", "ND", "NE",
            "NH", "NJ", "NM", "NV", "NY", "OH", "OK", "OR", "PA", "RI",
            "WY"};

        listBox1->Items->Clear();
        listBox1->Items->AddRange(states);
        listBox1->Height = 100;
        listBox1->Width = 110;
        listBox1->Location = Point(10, 10);

        this->Controls->Add(listBox1);
        this->Size = Drawing::Size(150, 150);
    }
};

[STAThread]
int main() {
    Application::Run(gcnew Form1);
}
```

The output looks:

![Image of a ListBox with states from the US]
The ArrayList class

One of the annoying problems of array in C++ as well as many programming language is that array must have a fixed size. Once the size of an array is declared, it cannot change. The ArrayList class is a subclass of the System.Collections which allows to create an array whose size is dynamically increased as required. Microsoft uses the term capacity to describe the dynamic size of an ArrayList which is the number of elements the ArrayList can hold. As elements are added to an ArrayList, the capacity is automatically increased as required through reallocation.

To create an ArrayList object, it is necessary to use the System::Collections namespace as shown below.

```c++
using namespace System::Collections;
```

The following is a complete code illustrating how to create an ArrayList object without defining the size. While the Capacity property sets the number of elements that the ArrayList can contain, the Count property actually returns the number of elements actually contained in the ArrayList. Capacity is always greater than or equal to Count. If Count exceeds Capacity while adding elements, the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements. The capacity grows in \(2^n\). It starts with 4, advances to 8, 16, and so on.

```c++
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;
using namespace System::Collections;

int main()
{
    ArrayList^ course = gcnew ArrayList;
    course->Add("CIS211");
    course->Add("CIS212");
    course->Add("CIS213");

    String^ str = "Capacity: " + course->Capacity + "\n";
    str += "Count: " + course->Count + "\n";

    MessageBox::Show(str);
}
```

Elements in this collection can be accessed using an integer index. Indexes in this collection are zero-based. For example,

```c++
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;
using namespace System::Collections;

int main()
{
    ArrayList^ course = gcnew ArrayList;
    course->Add("CIS211");
    course->Add("CIS212");
    course->Add("CIS213");

    String^ str;
    for (int i=0; i < course->Count; i++)
    {
        str += course[i] + "\n";
    }

    MessageBox::Show(str);
}
```
The `Add()` method adds an object to the end of the ArrayList. The `Clear()` method removes all elements from the ArrayList. The following uses a `for..each` loop to list all elements. It also demonstrates how the `Clear()` method works.

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;
using namespace System::Collections;

int main()
{
    ArrayList^ course = gcnew ArrayList;
    course->Add("CIS211");
    course->Add("CIS212");
    course->Add("CIS213");

    String^ str = "Capacity: " + course->Capacity + "\n";
    str += "Count: " + course->Count + "\n";

    for each(String^ c in course)
    {
        str += c + "\n";
    }

    MessageBox::Show(str);

    course->Add("CIS214");
    course->Add("CIS215");

    str = "New list\n";
    str += "Capacity: " + course->Capacity + "\n";
    str += "Count: " + course->Count + "\n";

    for each(String^ c in course)
    {
        str += c + "\n";
    }

    MessageBox::Show(str);

    course->Clear();
    course->TrimToSize();

    str = "After clearance\n";
    str += "Capacity: " + course->Capacity + "\n";
    str += "Count: " + course->Count + "\n";

    MessageBox::Show(str);
}
```

The output looks:
The capacity can be decreased by calling `TrimToSize` or by setting the `Capacity` property explicitly.

```c++
course->Capacity = 2;
```

By the way, using multidimensional arrays as elements in an ArrayList collection is not supported.

Interestingly, the ArrayList collection accepts “nullptr” as a valid value, allows duplicate elements; therefore, ArrayList may not always offer the best performance for a given task. Microsoft warns that ArrayList may not always offer the best performance for a given task. Microsoft recommends the List<T> class which will be discussed later.

The List<T> class

The List<T> class is the generic equivalent of the ArrayList class. It implements the IList<T> generic interface using an array whose size is dynamically increased as required. According to Microsoft, for very large List<T> objects, you can increase the maximum capacity to 2 billion elements on a 64-bit system by setting the enabled attribute of the gcAllowVeryLargeObjects configuration element to true in the run-time environment.

To create a List<T> object, it is necessary to use the System::Collections::Generic namespace as shown below.

```c++
using namespace System::Collections::Generic;
```

The following is the complete code that demonstrates how List<T> work. This code is very similar to the one for ArrayList. Similar to the ArrayList, the Add() method adds an object to the end of the List<T>. The Clear() method removes all elements from the List<T>.

```c++
#include <System.dll>
#include <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;
using namespace System::Collections::Generic;

int main()
{
    List<String^>^ city = gcnew List<String^>();
    String^ str = "Capacity: " + city->Capacity + "\n\n";
    city->Add("Tokyo");
    city->Add("Taipei");
    city->Add("Singapore");
    city->Add("Hong Kong");
    city->Add("Beijing");
    for each(String^ c in city)
    {
You can use the `Insert()` method to insert an element into the `List<T>` at the specified index. The syntax is:

```
Insert(index, item)
```

For example,

```c
    city->Insert(2, "New Delhi");
```

Microsoft said the `List<T>` class performs better in most cases and is type safe. If a reference type is used for type `T` of the `List<T>` class, the behavior of the two classes is identical. However, if a value type is used for type `T`, you need to consider implementation and boxing issues.

The following code uses a while loop to continuously add new contents:

```c
#using <System.dll>
#using <System.Windows.Forms.dll>
#include "InputBox.cpp"
using namespace System;
using namespace System::Windows::Forms;
using namespace System::Collections::Generic;

int main()
{
    List<String^>^ city = gcnew List<String^>();
    String^ str = "Capacity: " + city->Capacity + "\n"
    MessageBox::Show(str);
    str => "The new list is:";
    for each(String^ dinosaur in city)
    {
        str += dinosaur + "\n";
    }
    str += "Count: " + city->Count + "\n";
    MessageBox::Show(str);
}
```

str += "\nCapacity: " + city->Capacity + "\n";
str += "Count: " + city->Count + "\n";

MessageBox::Show(str);

while(true)
{
    // asking for new element
    city->Add(InputBox::Show("Enter one more city in Asia: "));
    str = "Capacity: " + city->Capacity + "\n";
    str += "Count: " + city->Count + "\n";

    for each(String^ c in city)
    {
        str += c + "\n";
    }
    MessageBox::Show(str);

    String^ ans = InputBox::Show("Add more? [Y/N] ");
    Char c = ans[0];
    if (c=='N' || c=='n')
    {
        break;
    }
}

Questions
1. Which is a jagged array in Visual C++?
A. array<array<int>^>^ a = gcnew array<array<int> ^>(3, 2);
B. array<int^>^ a = gcnew array<int ^>[3,2];
C. array<array<int>^>^ a = gcnew array<array<int> ^>(3);
D. array<int^>^ a = gcnew array<int ^>[3][2];

2. Given the following array, which represents 4?
array<int^>^ n = gcnew array<int ^>{ 1, 2, 3, 4, 5 };
A. n[0]
B. n[1]
C. n[2]
D. n[3]

3. Given the following array, what is the output of n[2] + n[3]?
array<int^>^ n = gcnew array<int ^>{ 1, 2, 3, 4, 5 };
A. 2 + 3
B. 5
C. 3 + 4
D. 7

4. Which is a valid array declaration in C++?
A. int^ n[5] = {1, 2, 3, 4, 5};
B. array<double ^> ^ a = gcnew array<double ^>[5];
C. array<int ^> ^ n = gcnew array<double ^> {1, 2, 3, 4, 5};
D. array<double ^> ^ a = gcnew array<double ^>(5) {1.1, 2.2, 3.3, 4.4, 5.5};

5. Which is the correct way to create a managed array with 5 elements?
A. array<double ^> ^ a(5) = gcnew array<double ^>;
B. array<double ^> ^ a = gcnew array<double ^>[5];
C. array<double ^> ^ a = gcnew array<double ^>(5);
D. array<double ^> ^ a[5] = gcnew array<double ^>;

6. In C++, which can retrieve the number of elements of a managed array named "fruit"?
A. MessageBox::Show(fruit[Size] + "");
B. MessageBox::Show(fruit.Length + "");
C. MessageBox::Show(fruit->Size + "");
D. MessageBox::Show(fruit[Length] + "");

7. Given the following table that defines the values of elements of a two-dimensional array. Which can return the value "Chem"?

<table>
<thead>
<tr>
<th>element of array &quot;Subject&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>row\col</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

A. subject[0, 1]
B. subject[0, 2]
C. subject[2, 2]
D. subject[1, 1]

8. Given the following table that defines the values of elements of a two-dimensional array. Which can declare and instantiate the array?

<table>
<thead>
<tr>
<th>element of array &quot;Subject&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>row\col</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

A. array<int, 2, 3>^ subject = gcnew array<int, 2, 3>;
B. array<int, 2>^ subject = gcnew array<int, 2>(2,3);
C. array<2, 3>^ subject = gcnew array<2, 3>;
D. array<int, 2>^ subject = gcnew array<int, 2>(3,2);

9. The output of the following code is __.

    array<String^, 3> ^ clr = gcnew array<String^, 3>(1, 4, 2);
    MessageBox::Show(A->Rank + "");

A. 1
B. 2
C. 3
D. 4

10. Given a three-dimensional array "n" and a single dimensional array "length" which calculate the number of elements of row, column, and layer of n, as shown below. Which returns the value 4?

```cpp
int n[2][4][2];
int length[3];
length[2] = (sizeof(n[0][0])/sizeof(int));
```
length[1] = (sizeof(n[0])/sizeof(int)) / length[2];
length[0] = (sizeof(n)/sizeof(int)) / (sizeof(n[0])/sizeof(int));

A. int length[3];
B. length[2] = (sizeof(n[0][0])/sizeof(int));
C. length[1] = (sizeof(n[0])/sizeof(int)) / length[2];
D. length[0] = (sizeof(n)/sizeof(int)) / (sizeof(n[0])/sizeof(int));
**Visual C++**

Lab #11 Visual C++ Array

**Learning Activity #1: single-dimensional managed array**

1. Create the C:\cis223 directory if it does not exist.

2. Launch the Developer Command Prompt (not the regular Command Prompt) and change to the C:\cis223 directory.

3. Use Notepad to create a new text file named `lab11_1.cpp` with the following contents:

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;

int main()
{
    String^ str;

    // integer array
    array<int ^> ^ x = gcnew array<int ^>(4) {5, 2, 8, 3};

    for (int i=0; i<x->Length; i++)
    {
        str += x[i] + " ";
    }
    str += "n
n";

    // double array
    array<double ^> ^dbl = gcnew array<double ^>(3);
    dbl[0] = 6.51;
    dbl[1] = 3.14;
    dbl[2] = 0.97;

    for each(double j in dbl)
    {
        str += j + "\n";
    }

    // String array
    array<String ^> ^ clr = gcnew array<String ^(7);
    clr[0] = "Green";
    clr[1] = "Blue";
    clr[2] = "Yellow";
    clr[3] = "Purple";
    clr[4] = "Pink";
    clr[5] = "Red";
    clr[6] = "White";

    Random^ rn = gcnew Random();
    int i = rn->Next() % 7;
    str += "\nToday\'s lucky color is " + clr[i] + "!";

    MessageBox::Show(str);

    return 0;
}
```
4. Type `cl /clr lab11_1.cpp /link /subsystem:windows /ENTRY:main` and press [Enter] to compile. Test the program. A sample output looks:

![Sample Output](image)

5. Download the “assignment template” and rename it to `lab11.doc` if necessary. Capture a screen shot similar to the above figure and paste it to the Word document named `lab11.doc` (or `.docx`).

**Learning Activity #2: Jagged array**

1. Use Notepad to create a new text file named `lab11_2.cpp` with the following contents:

```cpp
#include <System.dll>
#include <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    Form^ form1 = gcnew Form;
    form1->Width=200;
    form1->Height=370;

    Label^ label1 = gcnew Label;
    label1->AutoSize = true;
    form1->Controls->Add(label1);

    // code
    String^ str = "Jagged array\n\nOfficial way\n";

    array<array<int>^>^ a = gcnew array<array<int>^>(3);

    // 1st element of a
    a[0] = gcnew array<int>(4);
    a[0][0] = 1;
    a[0][1] = 2;
    a[0][2] = 3;
    a[0][3] = 4;

    // 2nd element of a
    a[1] = gcnew array<int>(2);
    a[1][0] = 5;
    a[1][1] = 6;

    // 3rd element of a
    a[2] = gcnew array<int>(3);
    a[2][0] = 7;
    a[2][1] = 8;
    a[2][2] = 9;

    for (int i=0; i < a->Length; i++)
    {
        for (int j=0; j < a[i]->Length; j++)
            // code
    

    label1->Text = str + a[0][0] + " " + a[0][1] + " " + a[0][2] + " " + a[0][3] + " " + a[1][0] + " " + a[1][1] + " " + a[2][0] + " " + a[2][1] + " " + a[2][2] + " " + a[2][3];
    
    label1->AutoSize = true;
}
```
{ 
    str += a[i][j] + "\n";
}
}

str += "\nShort-hand way\n";

array<array<String^>^>^ food = gcnew array<array<String^>^>(3)
{
    ["apple", "banaba", "orange"],
    ["onion", "pepper"],
    ["broccoli", "potato", "tomato", "spinach"]
};

for (int i=0; i < food->Length; i++)
{
    for (int j=0; j < food[i]->Length; j++)
    {
        str += food[i][j] + "\n";
    }

    // display the content
    label1->Text = str;
    Application::Run(form1);
}

2. Type cl /clr lab11_2.cpp /link /subsystem:windows /ENTRY:main and press [Enter] to compile. Test the program. A sample output looks:

3. Capture a screen shot similar to the above figure and paste it to the Word document named lab11.doc (or .docx).

Learning Activity #3: two-dimensional arrays

1. Use Notepad to create a new text file named lab11_3.cpp with the following contents:

```cpp
#include <System.dll>
#include <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main() {
```
String^ str = "\n2D Managed array:\n";

// short-hand
array<int, 2>^ A = gcnew array<int, 2>(2,4) {{53,97,46,81},{77,64,82,68}};

for (int i=0; i < A->GetLength(0); i++) { // row
    for (int j=0; j < A->GetLength(1); j++) { // column
        str += "A[" + i + "," + j + "] = " + A[i,j] + \\
    }
}
str += "\n";

// official
array<int, 2>^ B = gcnew array<int, 2>(2,4);
B[0,0] = 53;
B[0,1] = 97;
B[0,2] = 46;
B[0,3] = 81;
B[1,0] = 77;
B[1,1] = 64;
B[1,2] = 82;
B[1,3] = 68;

for (int i=0; i < B->GetLength(0); i++)
{
    for (int k=0; k < B->GetLength(0); k++)
    {
        str += "B[" + i + "," + k + "] is " + B[i,k] + "\n";
    }
}
str += "\n";

array<String^, 2>^ fruit = gcnew array<String^, 2>(2,3);
fruit[0,0] = "apple";
fruit[0,1] = "orange";
fruit[0,2] = "banana";
fruit[1,0] = "tangerine";
fruit[1,1] = "grape";
fruit[1,2] = "watermelon";

for (int i=0; i < fruit->GetLength(0); i++)
{
    for (int j=0; j < fruit->GetLength(0); j++)
    {
        str += "fruit[" + i + "," + j + "] is " + fruit[i,j] + "\n";
    }
}
str += "\n";

array<String^, 2>^ food = gcnew array<String^, 2>(3,2) {{"tofu", "sushi"},
{"cookie", "bread"},{"milk", "juice"}};

for each (String^ m in food)
{
    str += m + "\n";
}
MessageBox::Show(str);
return 0;
}

2. Compile and test the program. A sample output looks:
3. Capture a screen shot similar to the above figure and paste it to the Word document named lab11.doc (or .docx).

Learning Activity #4: Three-dimensional array

1. Use Notepad to create a new text file named lab11_4.cpp with the following contents:

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>
#using <System.Drawing.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;

public ref class Form1: public Form {

public:
    Form1() {
        // Managed array
        array<String^, 3> ^ clr = gcnew array<String^, 3>(2, 4, 2);
        clr[0, 0, 0] = "red";
        clr[0, 0, 1] = "green";
        clr[0, 1, 0] = "blue";
        clr[0, 1, 1] = "orange";
        clr[0, 2, 0] = "black";
        clr[0, 2, 1] = "gold";
        clr[0, 3, 0] = "beige";
        clr[0, 3, 1] = "silver";
        clr[1, 0, 0] = "yellow";
        clr[1, 0, 1] = "pink";
        clr[1, 1, 0] = "teal";
        clr[1, 1, 1] = "white";
        clr[1, 2, 0] = "gray";
        clr[1, 2, 1] = "cyan";
        clr[1, 3, 0] = "brown";
        clr[1, 3, 1] = "lime";

        String^ str1 = "Managed array" + Environment::NewLine;
    }

};
```
for (int i = 0; i < clr->GetLength(0); i++) {
    for (int j = 0; j < clr->GetLength(1); j++) {
        for (int k = 0; k < clr->GetLength(2); k++) {
            str1 += "clr[" + i + ", " + j + ", " + k + "] is " + clr[i, j, k] + Environment::NewLine;
        }
    }
}

// native array
int n[2][4][2] = {
    {{21, 37}, {16, 9}, {54, 83}, {17, 60}},
    {{44, 98}, {17, 56}, {29, 67}, {47, 18}},
};

int length[3];
length[2] = (sizeof(n[0][0])/sizeof(int));
length[1] = (sizeof(n[0])/sizeof(int)) / length[2];
length[0] = (sizeof(n)/sizeof(int)) / (sizeof(n[0])/sizeof(int));

String^ str2 = "Native array" + Environment::NewLine;

for (int i = 0; i < length[0]; i++) {
    for (int j = 0; j < length[1]; j++) {
        for (int k = 0; k < length[2]; k++) {
            str2 += "n[" + i + "][" + j + "][" + k + "] is " + n[i][j][k] + Environment::NewLine;
        }
    }
}

TextBox^ textBox1 = gcnew TextBox;
textBox1->Text = str1;
textBox1->Multiline = true;
textBox1->AutoSize = true;
textBox1->Location = Point(10, 10);
textBox1->Size = Drawing::Size(120, 240);
this->Controls->Add(textBox1);

TextBox^ textBox2 = gcnew TextBox;
textBox2->Text = str2;
textBox2->Multiline = true;
textBox2->AutoSize = true;
textBox2->Location = Point(150, 10);
textBox2->Size = Drawing::Size(120, 240);
this->Controls->Add(textBox2);
}

[STAThread]
int main() {
    Application::Run(gcnew Form1);
}

2. Test the program. A sample output looks:
3. Capture a screen shot similar to the above figure and paste it to the Word document named lab11.doc (or .docx).

Learning Activity #5: ArrayList and List<T>
1. Use Notepad to create a new text file named lab11_5.cpp with the following contents:

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>
#using <System.Drawing.dll>
using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;
using namespace System::Collections;
using namespace System::Collections::Generic;

public ref class Form1: public Form
{
    Label^ arr;
    Label^ lst;
    TextBox^ addArr;
    TextBox^ addLst;

    ArrayList^ course; // declare ArrayList
    List<String^>^ city; // declare List<T>

public:
    Form1() {
        this->Width = 530;

        arr = gcnew Label; // display contents of ArrayList
        arr->Text = "ArrayList:\n";
        arr->Width = 200;
        arr->Height = 150;
        arr->Location = Point(10, 50);
        Controls->Add(arr);

        lst = gcnew Label; // display content of List<T>
        lst->Text = "List<T>:\n";
        lst->Width = 200;
        lst->Height = 150;
        lst->Location = Point(220, 50);
        Controls->Add(lst);

        addArr = gcnew TextBox; // enter new value to ArrayList
        addArr->Width = 200;
        addArr->Location = Point(10, 10);
        Controls->Add(addArr);
    }
};
```
addLst = gcnew TextBox; // enter new values to List<T>
addLst->Width = 200;
addLst->Location = Point(220, 10);
Controls->Add(addLst);

Button^ button1 = gcnew Button;
button1->Text = "Add";
button1->Location = Point(430, 10);
button1->Click += gcnew EventHandler(this, &Form1::button1_Click);
Controls->Add(button1);

course = gcnew ArrayList; // initialize ArrayList
city = gcnew List<String^>(); // initialize List<T>
}

private: Void button1_Click(Object^ sender, EventArgs^ e) {
    // ArrayList
    if (addArr->Text != "")
    {
        course->Add(addArr->Text); // add new element to ArrayList
        String^ arrStr = "ArrayList:\n";
        for each(String^ c in course) // retrieve all elements of ArrayList
        {
            arrStr += c + "\n";
        }
        arrStr += "Capacity: " + course->Capacity + "\n";
        arrStr += "Count: " + course->Count + "\n"; // total number of element
        arr->Text = arrStr; // display the new content
        addArr->Text = ""; // clear the textbox
    }
    // List<T>
    if (addLst->Text != "")
    {
        city->Add(addLst->Text); // add new element to List<T>
        String^ arrLst = "List<T>:\n";
        for each(String^ c in city) // retrieve all elements of List<T>
        {
            arrLst += c + "\n";
        }
        arrLst += "Capacity: " + city->Capacity + "\n";
        arrLst += "Count: " + city->Count + "\n"; // total number of element
        lst->Text = arrLst; // display the new content
        addLst->Text = ""; // clear the textbox
    }
}

[STAThread]
int main()
{
    Application::Run(gcnew Form1);
}

2. Compile and test the program. A sample output looks:
Visual C++ Programming – Penn Wu, PhD

3. Capture a screen shot similar to the above figure and paste it to the Word document named lab11.doc (or .docx).

Submittal
1. Complete all the 5 learning activities and the programming exercise in this lab.

2. Create a .zip file named lab11.zip containing ONLY the following self-executable files.
   - Lab11_1.exe
   - Lab11_2.exe
   - Lab11_3.exe
   - Lab11_4.exe
   - Lab11_5.exe
   - lab11.doc (or lab11.docx or .pdf) [You may be given zero point if this Word document is missing]

3. Upload the zipped file to Question 11 of Assignment as response.

Programming Exercise 11:
1. Launch the Developer Command Prompt.

2. Use Notepad to create a new text file named ex11.cpp.

3. Add the following heading lines (Be sure to use replace [YourFullNameHere] with the correct one).

   //File Name: ex11.cpp
   //Programmer: [YourFullNameHere]

4. Hand code a Windows form application that contains a two-dimensional array, as specified by the following table:

<table>
<thead>
<tr>
<th>Column\Row</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Kyoto</td>
<td>Chikuba</td>
<td>Hiroshima</td>
</tr>
<tr>
<td>1</td>
<td>Osaka</td>
<td>Nagasaki</td>
<td>Sapporo</td>
</tr>
</tbody>
</table>

5. Use two *for* loops to display each of the element’s value in a message box, as shown below.
6. Your application must meet the above requirements to get credit. No partial credit is given.

7. Download the “programming exercise template” and rename it to ex11.doc if necessary. **Copy your source code to the file** and then capture the screen shot(s) similar to the above one and paste it to the Word document named “ex11.doc” (or .docx).


**Grading criteria:**
You will earn credit only when the following requirements are fulfilled. No partial credit is given.
- You successfully submit both source file and executable file.
- Your source code must be fully functional and may not contain syntax errors in order to earn credit.
- Your executable file (program) must be executable to earn credit.

**Threaded Discussion**
**Note:** Students are required to participate in the thread discussion on a weekly basis. Student must post at least two messages as responses to the question every week. Each message must be posted on a different date. Grading is based on quality of the message.

**Question:** Class, in your opinion, what is/are the limit(s) or constraint(s) that keep(s) a programmer from using a Visual C++ array in application that display all the contact information of customers? What will be your solution(s) if there is/are limit(s) or constraint(s)? If you believe that there is not limit/constraint, then explain why? [There is never a right-or-wrong answer for this question. Please free feel to express your opinion.]

   Be sure to use proper college level of writing. Do not use texting language.