Experiment 9
Templates

Objectives

To use function templates to conveniently create a group of related (overloaded) functions.
To distinguish between function templates and function-template specializations.
To use class templates to create groups of related types.
To distinguish between class templates and classtemplate specializations.
To overload function templates.
To understand the relationships among templates, friends, inheritance and static members.

Prelab Activities

Programming Output

For each of the given program segments, read the code and write the output in the space provided below each program. [Note: Do not execute these programs on a computer.]

1. What is output by the following program?

```
1  #include <iostream>
2  using namespace std;
3
4  // function template mystery definition
5  template< typename T >
6  void mystery( const T *a, const int c )
7  {
8     for ( int i = 0; i < c; i++ )
9     cout << a[ i ] << " ** ";
10
11    cout << endl;
12 } // end function mystery
13
14 int main()
15 {
16    const int size = 5;
17
18    int i[ size ] = { 22, 33, 44, 55, 66 };
19    char c[ size ] = { 'c', 'd', 'g', 'p', 'q' };
20
21    mystery( i, size );
22    cout << endl;
23    mystery( c, size - 2 );
24    cout << endl;
25 } // end main
```

2. What is output by the following program?

```
#include <iostream>
#include <new>
#include <iomanip>
using namespace std;

// class template for class Array
template< typename T >
class Array
{
public:
   Array( int = 5 );
   ~Array() { delete [] arrayPtr; }
   T arrayRef( int ) const;
   int getSize() const;
private:
   int size;
   T *arrayPtr;
}; // end class Array

// constructor for class Array
template< typename T >
Array< T >::Array( int x )
{
   size = x;
   arrayPtr = new T[ size ];

   for ( int i = 0; i < size; i++ )
     arrayPtr[ i ] = 1.0 * i;
} // end class Array constructor

// function arrayRef definition
template< typename T >
T Array< T >::arrayRef( int num ) const
{
   return arrayPtr[ num ];
} // end function arrayRef

// return size
template< typename T >
   int Array< T >::getSize() const
   {
      return size;
   } // end function getSize

  //non-member function template to print an object of type Array
template< typename T >
void printArray( const Array< T > &a )
{
   for ( int i = 0; i < a.getSize(); i++ )
   cout << a.arrayRef( i ) << " ";

   cout << endl << endl;
} // end function printArray

```

```
55 int main()
56 {
57    Array< int > intArray( 4 );
58    Array< double > doubleArray;
59
60    cout << setprecision( 2 ) << fixed;
61    printArray( intArray );
62    printArray( doubleArray );
63 } // end main
```

CorrecttheCode

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic or compilation error, write the corrected code. If the code does not contain an error, write "no error." For code segments, assume the code appears in main and that using directives are provided. [Note: It is possible that a program segment may contain multiple errors.]

1. The following code invokes the function print:

```
1  #include <iostream>
2  using namespace std;
3
4  // template function print definition
5  template < class T >
6  void print( T left, T right )
7  {
8     cout << "Printing arguments: " << left
9          << " ** " << right;
10 } // end function print
11
12 int main()
13 {
14    cout << endl;
15    print( 3, 5.8 );
16    cout << endl;
17 } // end main
```

2. The following is a class definition for a class template Stack:

```
1  #ifndef TSTACK_H
2  #define TSTACK_H
3
4  // template class Stack definition
5  template< class T >
6  class Stack
7  {
8  public:
9     Stack( int = 10 );
10
11    // destructor
12    ~Stack()
13    {
14       delete [] stackPtr;
15
16    } // end class Stack destructor
17
18    bool push( const T& );
```

```
19    bool pop( T& );
20 private:
21    int size;
22    int top;
23    T *stackPtr;
24
25    // function isEmpty definition
26    bool isEmpty() const
27    {
28       return top == -1;
29
30    } // end function isEmpty
31
32    // function isFull definition
33    bool isFull() const
34    {
35       return top == size - 1;
36
37    } // end function isFull
38 }; // end class Stack
39
40 // constructor
41 Stack< T >::Stack( int s )
42 {
43    size = s > 0 ? s : 10;
44    top = -1;
45    stackPtr = new T[ size ];
46 } // end class Stack constructor
47
48 // function push definition
49 bool Stack< T >::push( const T &pushValue )
50 {
51    if ( !isFull() )
52    {
53       stackPtr[ ++top ] = pushValue;
54       return true;
55    } // end if
56
57    return false;
58 } // end function push
59
60 // function pop definition
61 bool Stack< T >::pop( T &popValue )
62 {
63    if ( !isEmpty() )
64    {
65       popValue = stackPtr[ top-- ];
66       return true;
67    } // end if
68
69    return false;
70 } // end function pop
71
72 #endif // TSTACK H
```

3. The following code invokes function print:

```
1  #include <iostream>
2  using namespace std;
3
4  // class MyClass definition
5  class MyClass
6  {
7  public:
8     MyClass();
9
10    void set( int );
11    int get() const;
12 private:
13    int data;
14 }; // end class MyClass
15
16 // constructor
17 MyClass::MyClass()
18 {
19    set( 0 );
20 } // end class MyClass constructor
21
22 // function setData definition
23 void MyClass::setData( int num )
24 {
25    data = num >= 0 ? num : -1;
26 } // end function setData
27
28 // return data
29 int MyClass::getData() const
30 {
31    return getData;
32 } // end function getData
33
34 // template function print
35 template < class T >
36 void print( T left, T right )
37 {
38    cout << "Printing arguments: " << left
39         << " ** " << right;
40 } // end function print
41
42 int main()
43 {
44    MyClass m1;
45    MyClass m2;
46
47    m1.setData( 2 );
48    m2.setData( 7 );
49
50    cout << endl;
51    print( m1, m2 );
52    cout << endl;
53 } // end main
```

Lab Exercises

Lab Exercise 1 — ArrayHelper

The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. UML Diagram
4. Sample Output
5. Test Code Template
6. Problem-Solving Tips

The test program template represents a complete working C++ test program. Read the problem description and examine the sample output; then study the test code. Using the problem-solving tips as a guide write your C++ code. Compile and execute the program. Compare your output with the sample output provided.

Lab Objectives

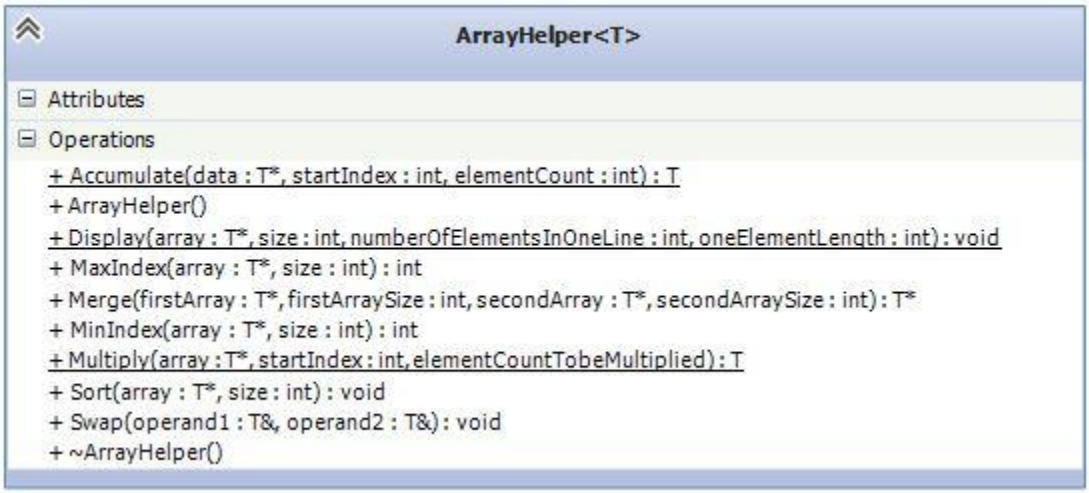
In this lab, you will practice:

- Creating a class using template to perform some basic array operations.
- Using function templates to create function-template specializations.

Description of the Problem

Create a class called ArrayHelper and use templates to create a generic implementation. By the help of the templates, this class can handle the all of the primitive types and user defines types after some modifications on the user defines types. In the test code, you will see an integer case of the code. You have to write test code for the case of double and character.

UML Diagram



Functions:

Display(array, size, numberofElementsInOneLine, onElementLength) : Displays the given array according to the 3rd and 4th argument.

MaxIndex(array,size) : Finds the maximum element and returns the index of it.

MinIndex(array,size) : Finds the minimum element and returns the index of it.

Merge(firstArray, firstArraySize, secondArray, secondArraySize) : Merges the given arrays and returns a new array containing the merged elements.

Sort(array, size) : Sorts the given array in ascending order.

Swap(operand1,operand2) : Swaps the given elements.

Accumulate(data, startIndex, elementCount): This function is resposible to accumulate(sum) the elements of array starting from the given "startIndex" to "elementCountTobeAccumulated" th element.

Multiply(data, startIndex, elementCountTobeMultiplied): This function is resposible to Multiply the elements of the array starting from the given "startIndex" to the "elementCountTobeMultiplied" th element.

Sample Output

```
IntegerArray1 Content:
Displaying the container
  17,  19, -12,  -6, -17
  16, -34, -49,  17,  19
IntegerArray1 Content:
Displaying the container
 -49, -34, -17, -12,  -6
  16,  17,  17,  19,  19
Min : -49
Max : 19
IntegerArray2 Content:
Displaying the container
  -2,  33, -12,   3,  44
Merged Array Content:
Displaying the container
 -49, -34, -17, -12,  -6
  16,  17,  17,  19,  19
  -2,  33, -12,   3,  44
integerData Content:
Displaying the container
     1,     2,     3,     4,     5
     6,     7,     8,     9,    10
Accumulate Result : 27
Multiply Result : 5040
Press any key to continue . . .
```

Test Code Template

```
/**************************************
**************************************
* IDE : Visual Studio 2015           *
* Author : Cihan UYANIK              *
* Experiment 7: Templates            *
**************************************/

#include  "ArrayHelper.h"
#include "Date.h"
#include <time.h>

template<class T>
T GenerateRandomNumber(int lowerBound,int upperBound, double scaler = 1)
{
	lowerBound *= scaler;
	upperBound *= scaler;

	T randomData = lowerBound + rand() % (upperBound - lowerBound + 1);

	randomData /= scaler;

	return randomData;
}



int main()
{
	srand(time(NULL));
	int minIndex, maxIndex;
	ArrayHelper<int> IntegerArrayHelper;

	int IntegerArray1[10];
	for (size_t i = 0; i < 10; i++)
	{
		IntegerArray1[i] = GenerateRandomNumber<int>(-50, 50);
	}

	cout << "IntegerArray1 Content:" << endl;
	ArrayHelper<int>::Display(IntegerArray1, 10, 5, 4);

	IntegerArrayHelper.Sort(IntegerArray1, 10);
	cout << "IntegerArray1 Content:" << endl;
	ArrayHelper<int>::Display(IntegerArray1, 10, 5, 4);

	minIndex = IntegerArrayHelper.MinIndex(IntegerArray1, 10);
	maxIndex = IntegerArrayHelper.MaxIndex(IntegerArray1, 10);
	cout << "Min : " << IntegerArray1[minIndex] << endl;
	cout << "Max : " << IntegerArray1[maxIndex] << endl;

	int IntegerArray2[5];
	for (size_t i = 0; i < 5; i++)
	{
		IntegerArray2[i] = GenerateRandomNumber<int>(-50, 50);
	}
```

```
        cout << "IntegerArray2 Content:" << endl;
        ArrayHelper<int>::Display(IntegerArray2, 5, 5, 4);

        int* newArray = IntegerArrayHelper.Merge(IntegerArray1, 10, IntegerArray2, 5);
        cout << "Merged Array Content:" << endl;
        ArrayHelper<int>::Display(newArray, 15, 5, 4);

        int integerData[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        cout << "integerData Content:" << endl;
        ArrayHelper<int>::Display(integerData, 10, 5, 6);

        int result = ArrayHelper<int>::Accumulate(integerData, 1, 6);
        cout << "Accumulate Result : " << result << endl;

        result = ArrayHelper<int>::Multiply(integerData, 1, 6);
        cout << "Multiply Result : " << result << endl;

        /*
        REPEAT SAME TEST FOR DOUBLE ARRAY
        AND CHARACTER ARRAY

        return 0;
}
```

Problem Solving Tips

1- Use UML Diagram, function definitions and sample output

Lab Exercise 2 — ArrayHelper with user defined types

The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. UML Diagram
4. Sample Output
5. Test Code
6. Problem-Solving Tips

The test program template represents a complete working C++ test program. Read the problem description and examine the sample output; then study the test code. Using the problem-solving tips as a guide write your C++ code. Compile and execute the program. Compare your output with the sample output provided.

Lab Objectives

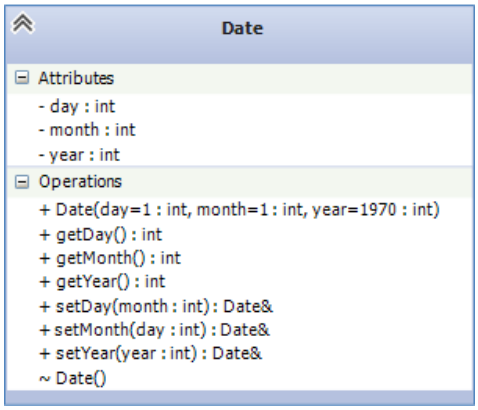
In this lab, you will practice:

- Creating a user defined type(class) to use in a template based algorithm.

Description of the Problem

Create a class called Date and create an array of type Date in the main. Then use this array as the container object to be passed to ArrayHelper object functions. If you have some type of errors depending on the absence of some functions in the user defined object(Date object), It means you need to implement these functions. Your job is to find out the required functions and to implement them. **You are not allowed to change the ArrayHelper class implemented in Lab. Exercise – I**

UML Diagram



Sample Output

```
Displaying the container
26-07-2009
26-06-2017
19-11-1976
04-02-1971
30-07-1976
22-02-2000
21-01-1990
02-10-1977
29-08-1991
21-12-2017
Displaying the container
04-02-1971
30-07-1976
19-11-1976
02-10-1977
21-01-1990
29-08-1991
22-02-2000
26-07-2009
26-06-2017
21-12-2017
Min : 04-02-1971
Max : 21-12-2017
Displaying the container
04-02-1971
30-07-1976
19-11-1976
02-10-1977
21-01-1990
29-08-1991
22-02-2000
26-07-2009
26-06-2017
21-12-2017
13-05-2012
12-12-2010
28-04-2016
05-03-2020
08-05-2005
Press any key to continue . . .
```

Test Code

```
/****************************************
*****************************************
* IDE : Visual Studio 2015              *
* Author : Cihan UYANIK                 *
* Experiment 7: Templates               *
*****************************************/

#include  "ArrayHelper.h"
#include "Date.h"
#include <time.h>

template<class T>
T GenerateRandomNumber(int lowerBound,int upperBound, double scaler = 1)
{
	lowerBound *= scaler;
	upperBound *= scaler;
	T randomData = lowerBound + rand() % (upperBound - lowerBound + 1);
	randomData /= scaler;
	return randomData;
}

int main()
{
	srand(time(NULL));
	int minIndex, maxIndex;

	Date DateArray1[10];
	for (size_t i = 0; i < 10; i++)
	{
		int day = GenerateRandomNumber<int>(1, 31);
		int month = GenerateRandomNumber<int>(1, 12);
		int year = GenerateRandomNumber<int>(1970, 2030);
		DateArray1[i] = Date(day, month, year);
	}
	ArrayHelper<Date> DateArrayHelper;
	ArrayHelper<Date>::Display(DateArray1, 10, 1, 20);
	DateArrayHelper.Sort(DateArray1, 10);
	ArrayHelper<Date>::Display(DateArray1, 10, 1, 20);
	minIndex = DateArrayHelper.MinIndex(DateArray1, 10);
	maxIndex = DateArrayHelper.MaxIndex(DateArray1, 10);
	cout << "Min : " << DateArray1[minIndex] << endl;
	cout << "Max : " << DateArray1[maxIndex] << endl;
	Date DateArray2[5];
	for (size_t i = 0; i < 5; i++)
	{
		int day = GenerateRandomNumber<int>(1, 31);
		int month = GenerateRandomNumber<int>(1, 12);
		int year = GenerateRandomNumber<int>(1970, 2030);
		DateArray2[i] = Date(day, month, year);
	}
	Date* newDateArray = DateArrayHelper.Merge(DateArray1, 10, DateArray2, 5);
	ArrayHelper<Date>::Display(newDateArray, 15, 1, 20);
	return 0;
}
```

Problem Solving Tips

1- Use UML diagram and test code given.
2- If you get an error as given in the following figure, it means you need to implement the output stream operator (<<) for the user defined type (class) “Date”.