**Lab Exercises**

Class: 软件1801 Name: 邓海燕 StuID:\_\_\_\_201816040130\_\_\_\_\_\_\_\_\_\_\_\_

Date: 2019.12.26 Score:

**Lab Exercise — Overloading printArray**

**I Lab Objectives**

In this lab, you will practice:

1. Overloading a function template for printing an array.
2. Using function templates to create function-template specializations.

**II Description of the Problem**

Overload function template printArray so that it takes two additional integer arguments, namely int lowSubscript and int highSubscript. A call to this function will print only the designated portion of the array. Validate lowSubscript and highSubscript; if either is out of range or if highSubscript is less than or equal to lowSubscript, the overloaded printArray function should return 0; otherwise, printArray should return the number of elements printed. Then modify main to demonstrate both versions of printArray on arrays a, b and c. Be sure to test all capabilities of both versions of printArray.

//Original program

// Using function-template specializations.

#include <iostream>

using namespace std;

// function template printArray definition

template< typename T >

void printArray( const T \* const array, int count )

{

for ( int i = 0; i < count; ++i )

cout << array[ i ] << " ";

cout << endl;

} // end function template printArray

int main()

{

const int aCount = 5; // size of array a

const int bCount = 7; // size of array b

const int cCount = 6; // size of array c

int a[ aCount ] = { 1, 2, 3, 4, 5 };

double b[ bCount ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };

char c[ cCount ] = "HELLO"; // 6th position for null

cout << "Array a contains:" << endl;

// call integer function-template specialization

printArray( a, aCount );

cout << "Array b contains:" << endl;

// call double function-template specialization

printArray( b, bCount );

cout << "Array c contains:" << endl;

// call character function-template specialization

printArray( c, cCount );

} // end main

**III Sample Output**



**IV Problem-Solving Tips**

1. To overload the printArray function template, declare another function template, also named printArray, that takes two additional int parameters, lowSubscript and highSubscript.
2. When iterating over the range from lowSubscript to highSubscript, make sure to include both values within the range, to avoid an off-by-one error.

**V Your Solution**

// Lab 1: TemplateOverload.cpp

// Using template functions

#include <iostream>

using namespace std;

// function template printArray definition

// original function

template< typename T >

void printArray( const T \*array, int count )

{

// display array

for ( int i = 0; i < count; i++ )

cout << array[ i ] << " ";

cout << endl;

} // end function printArray

// overloaded function template printArray

// takes upper and lower subscripts to print

template< typename T >

int printArray(const T \*array, int number ,int lowSubscrip,int highSubscript)

{

// check if subscript is negative or out of range

if ( lowSubscrip<0||highSubscript>number )

return 0;

int count = 0;

// display array

for (int i = lowSubscrip;i <= highSubscript;i++ )

{

++count;

cout << array[ i ] << ' ';

} // end for

cout << '\n';

return count; // number or elements output

} // end overloaded function printArray

int main()

{

const int ACOUNT = 5; // size of array a

const int BCOUNT = 7; // size of array b

const int CCOUNT = 6; // size of array c

// declare and initialize arrays

int a[ ACOUNT ] = { 1, 2, 3, 4, 5 };

double b[ BCOUNT ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };

char c[ CCOUNT ] = "HELLO"; // 6th position for null

int elements;

// display array a using original printArray function

cout << "\nUsing original printArray function\n";

printArray( a, ACOUNT );

// display array a using new printArray function

cout << "Array a contains:\n";

elements = printArray(a,ACOUNT,0,ACOUNT-1);

cout << elements << " elements were output\n";

// display elements 1-3 of array a

cout << "Array a from positions 1 to 3 is:\n";

elements = printArray(a,ACOUNT,1,3);

cout << elements << " elements were output\n";

// try to print an invalid element

cout << "Array a output with invalid subscripts:\n";

elements = printArray(a,5,-1,10);

cout << elements << " elements were output\n\n";

// display array b using original printArray function

cout << "\nUsing original printArray function\n";

printArray( b, BCOUNT );

// display array b using new printArray function

cout << "Array b contains:\n";

elements = printArray(b,7,0,BCOUNT-1);

cout << elements << " elements were output\n";

// display elements 1-3 of array b

cout << "Array b from positions 1 to 3 is:\n";

elements = printArray(b,7,1,3);

cout << elements << " elements were output\n";

// try to print an invalid element

cout << "Array b output with invalid subscripts:\n";

elements = printArray(b,7,-1,10);

cout << elements << " elements were output\n\n";

// display array c using original printArray function

cout << "\nUsing original printArray function\n";

printArray( c, CCOUNT );

// display array c using new printArray function

cout << "Array c contains:\n";

elements = printArray(c,6,0,CCOUNT-2);

cout << elements << " elements were output\n";

// display elements 1-3 of array c

cout << "Array c from positions 1 to 3 is:\n";

elements = printArray(c,6,1,3);

cout << elements << " elements were output\n";

// try to display an invalid element

cout << "Array c output with invalid subscripts:\n";

elements = printArray(c,6,-1,10);

cout << elements << " elements were output" << endl;

} // end main



