**Lab Exercises**

Class: 软件2102 Name: 刘京 StuID:\_211210400209\_

Date: 2022/12/18 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**

// LabExercises\_Templates.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

//Original program

// Using function-template specializations.

#include <iostream>

using namespace std;

// function template printArray definition

template< typename T >

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

{

int lowSubscript = 1;

int highSubscript = 3;

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

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

cout << endl;

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

if (highSubscript > lowSubscript) {

cout << "Array a from positions " << lowSubscript << " to " << highSubscript << " is:\n";

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

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

cout << endl;

cout << highSubscript - lowSubscript + 1 << " elements were output\n";

}

else

{

return 0;

}

} // 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 << "Using original printArray function\n";

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

cout << a[i] << " ";

cout << endl;

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

// call integer function-template specialization

printArray(a, aCount);

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

cout << "0 elements were output\n\n\n";

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

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

cout << b[i] << " ";

cout << endl;

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

// call double function-template specialization

printArray(b, bCount);

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

cout << "0 elements were output\n\n\n";

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

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

cout << c[i] << " ";

cout << endl;

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

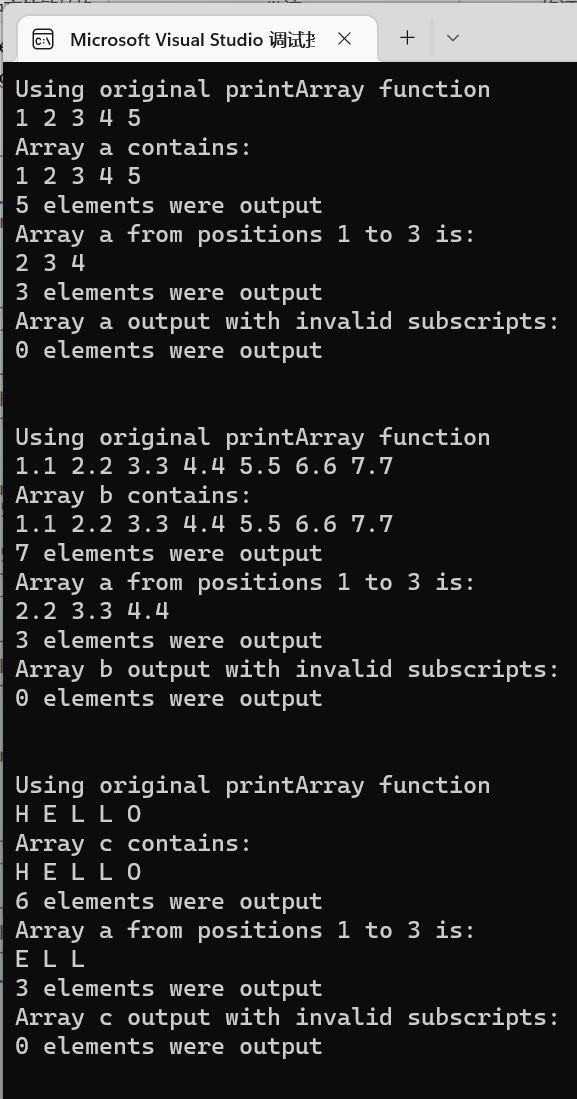
// call character function-template specialization

printArray(c, cCount);

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

cout << "0 elements were output\n\n\n";

} // end main

****