Object-Oriented Programming (Lab Exercise)
Spring 2024
Software Engineering Department, NED University of Engineering and Technology

Week #5

Objective:

The objective of this lab session is to introduce students to the concept of pointers in C++ and help them understand how to work with pointers effectively. By the end of this lab, students should be able to:

Understand the concept of pointers and their importance in C++ programming.
Declare, initialize, and dereference pointers.
Perform pointer arithmetic and understand pointer relationships.
Use pointers to work with arrays and dynamic memory allocation.
Identify common pitfalls and best practices when working with pointers.

Introduction:

In C++, a pointer is a variable that stores the memory address of another variable. Pointers are powerful tools that enable us to work with memory directly and perform operations such as dynamic memory allocation, passing parameters by reference, and building complex data structures like linked lists and trees.

Declaration and Initialization:

To declare a pointer variable, you use the asterisk (*) symbol followed by the data type of the variable it points to. Here's an example:

```
int *ptr;
```

This declares a pointer named ptr that can point to an integer variable.You can initialize a pointer with the address of another variable using the address-of operator (&):

```
int num = 10;
int *ptr = &num; // ptr now points to the memory address of num
```

Dereferencing:

Dereferencing a pointer means accessing the value stored at the memory address it points to. This is done using the asterisk (*) symbol. For example:

```
int num = 10;
int *ptr = &num;
cout << "Value of num: " << *ptr << endl; // Output: Value of num:
10
```

Here, *ptr retrieves the value stored at the memory address pointed to by ptr, which is 10.

Pointer Arithmetic:

Pointer arithmetic allows you to perform arithmetic operations on pointers. For example, you can increment or decrement a pointer, add an integer to a pointer, or subtract an integer from a pointer. Pointer arithmetic is often used when working with arrays and dynamic memory allocation.

```
int arr[] = {1, 2, 3, 4, 5};
int *ptr = arr; // Pointer to the first element of the array

cout << *ptr << endl; // Output: 1
ptr++; // Move to the next element
cout << *ptr << endl; // Output: 2
```

Null Pointers:

A null pointer is a pointer that does not point to any memory location. It is represented by the value nullptr in C++. Null pointers are often used to indicate that a pointer is not currently pointing to valid memory.

```
int *ptr = nullptr;
```

Dynamic Memory Allocation:

Pointers are commonly used for dynamic memory allocation using new and delete operators. Dynamic memory allocation allows you to allocate memory at runtime and is useful when the size of the memory needed is not known at compile time.

```
int *ptr = new int; // Allocate memory for an integer
*ptr = 10; // Assign a value to the memory location
delete ptr; // Free the allocated memory
```

Common Pitfalls:

Dangling Pointers: Pointers that point to memory that has been deallocated.

Memory Leaks: Failure to deallocate memory that was previously allocated dynamically.

Uninitialized Pointers: Using pointers that have not been initialized, leading to undefined behavior.

Understanding pointers is essential for mastering C++ programming and enables you to work with memory more efficiently. However, pointers require careful handling to avoid common pitfalls and memory-related issues.

Exercise:

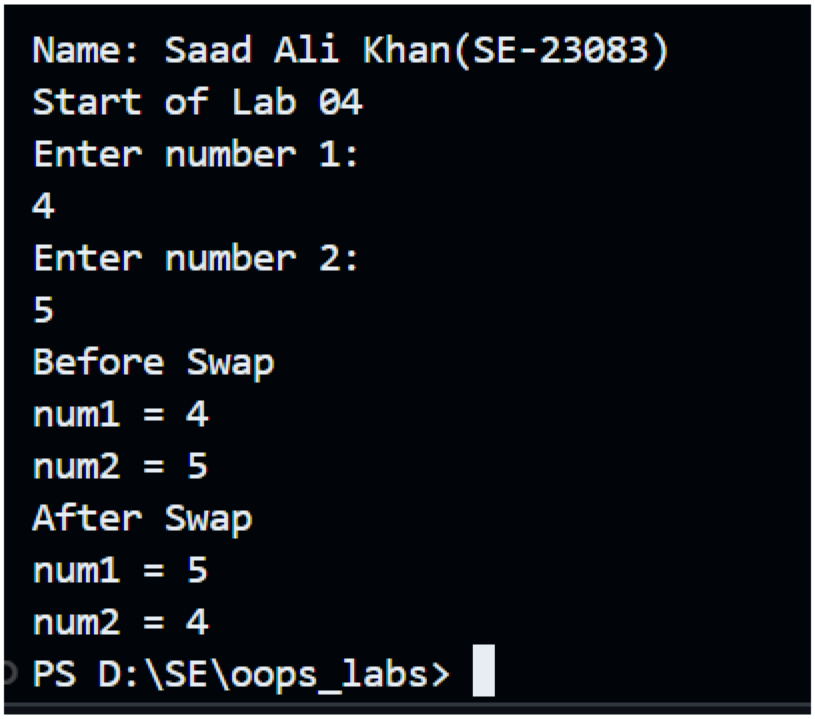
1. Write a function swap that takes two integer pointers as arguments and swaps the values they point to.

CODE:

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

int startlab1()
{
    cout << "Name: Saad Ali Khan(SE-23083)" << endl;
    cout << "Start of Lab 04" << endl;
    return 0;
}

int swap(int *ptr1, int *ptr2)
{
    int *temp;
    temp = ptr1;
    ptr1 = ptr2;
    ptr2 = temp;
    return 0;
}

int l4q1()
{
    int num1, num2;
    int *ptr1, *ptr2;
    cout << "Enter number 1: " << endl;
    cin >> num1;
    cout << "Enter number 2: " << endl;
    cin >> num2;

    ptr1 = &num1;
    ptr2 = &num2;
    cout << "Before Swap" << endl;
    cout << "num1 = " << *ptr1 << endl;
    cout << "num2 = " << *ptr2 << endl;
    swap(*ptr1, *ptr2);
    cout << "After Swap" << endl;
    cout << "num1 = " << *ptr1 << endl;
    cout << "num2 = " << *ptr2 << endl;
    return 0;
}

int main()
```

```cpp
{
    startlab1();
    l4q1();
    return 0;
}
```

OUTPUT:

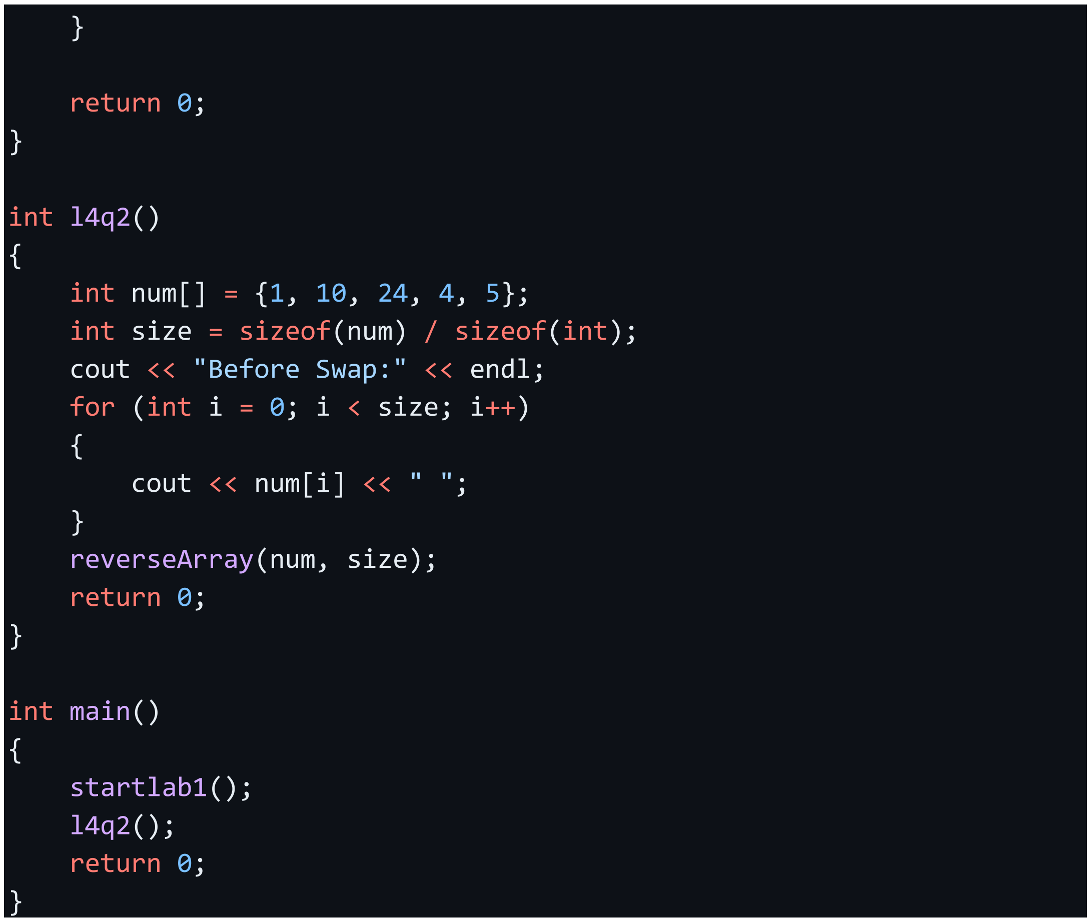


```
Name: Saad Ali Khan(SE-23083)
Start of Lab 04
Enter number 1:
4
Enter number 2:
5
Before Swap
num1 = 4
num2 = 5
After Swap
num1 = 5
num2 = 4
PS D:\SE\oops_labs>
```

2. Write a function reverseArray that takes an integer array and its size as arguments and reverses the elements of the array using pointers.

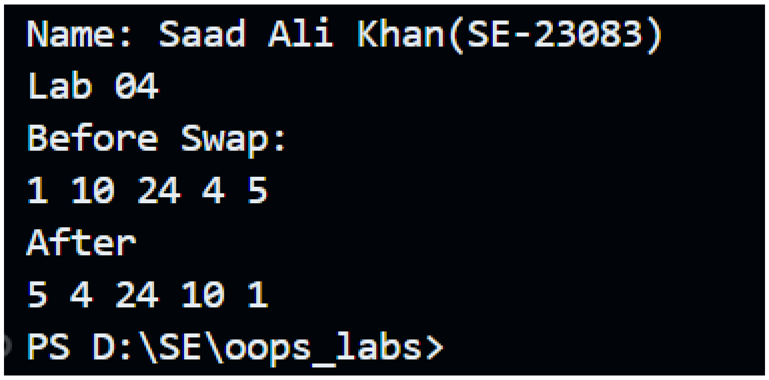
CODE:

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

int startlab1()
{
    cout << "Name: Saad Ali Khan(SE-23083)" << endl;
    cout << "Lab 04" << endl;
    return 0;
}

int reverseArray(int num[], int size)
{
    cout << endl;
    cout << "After" << endl;
    int *ptr = num;
    for (int i = size - 1; i >= 0; i--)
    {
        cout << ptr[i] << " ";
        ptr[i]--;
```



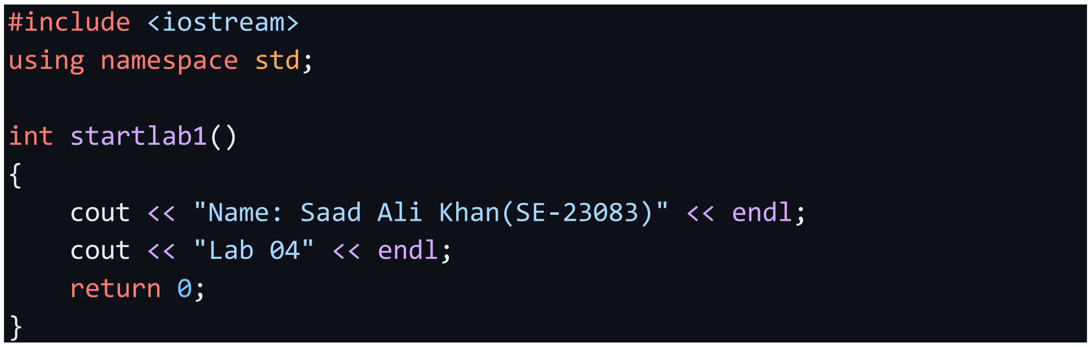
```
    }

    return 0;
}

int l4q2()
{
    int num[] = {1, 10, 24, 4, 5};
    int size = sizeof(num) / sizeof(int);
    cout << "Before Swap:" << endl;
    for (int i = 0; i < size; i++)
    {
        cout << num[i] << " ";
    }
    reverseArray(num, size);
    return 0;
}

int main()
{
    startlab1();
    l4q2();
    return 0;
}
```

CODE:

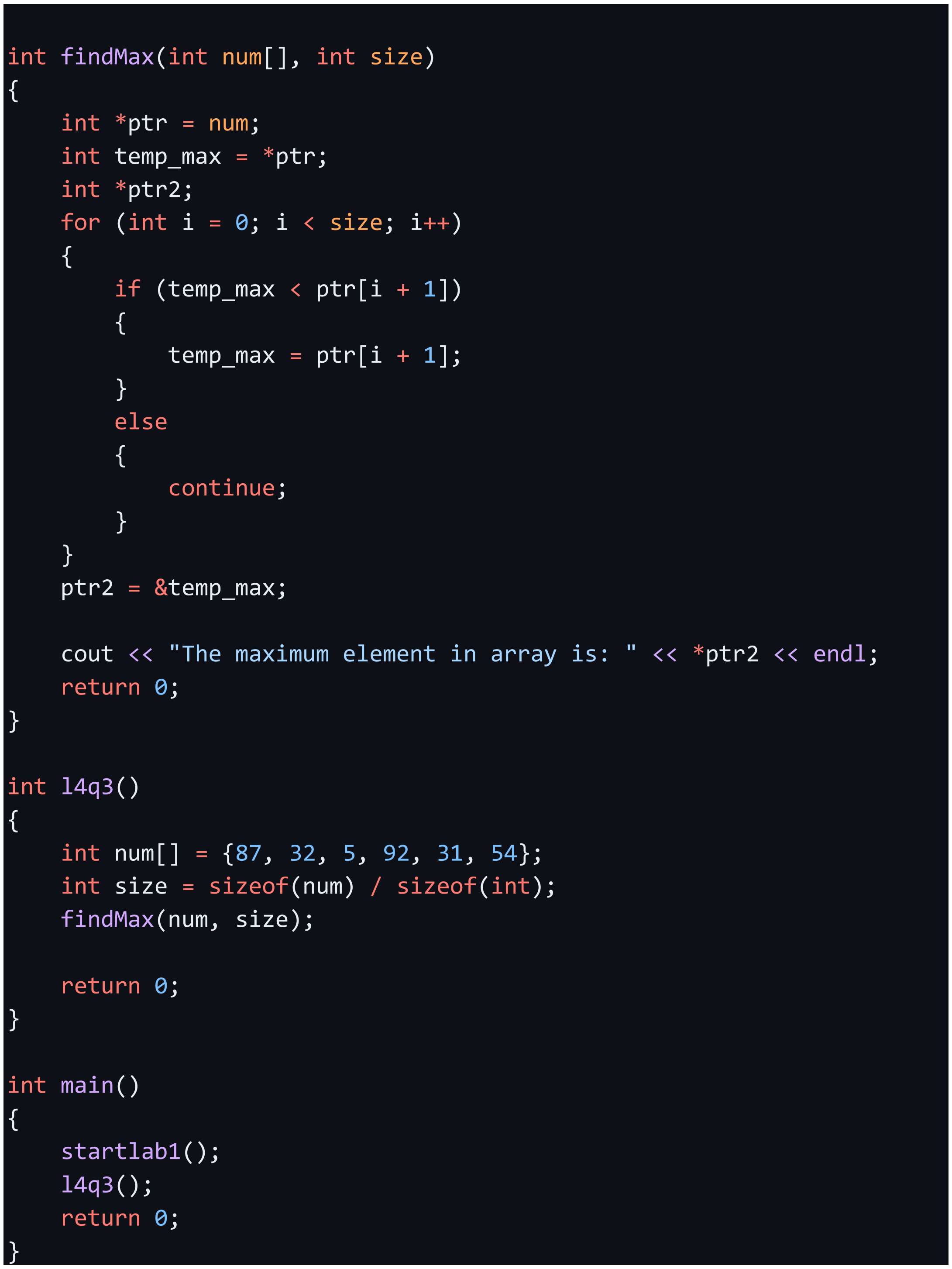


```
Name: Saad Ali Khan(SE-23083)
Lab 04
Before Swap:
1 10 24 4 5
After
5 4 24 10 1
PS D:\SE\oops_labs>
```

3. Write a function findMax that takes an integer array and its size as arguments and returns a pointer to the maximum element in the array.
CODE:

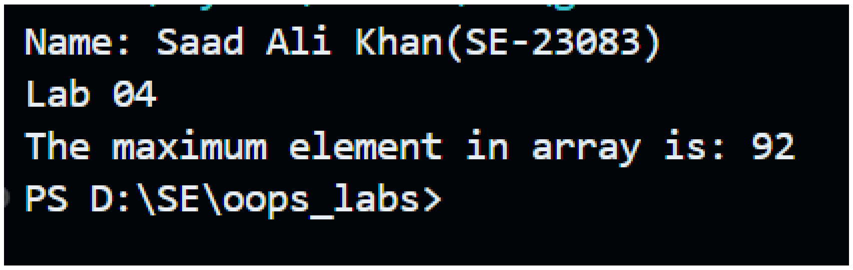


```
#include <iostream>
using namespace std;

int startlab1()
{
    cout << "Name: Saad Ali Khan(SE-23083)" << endl;
    cout << "Lab 04" << endl;
    return 0;
}
```



```
int findMax(int num[], int size)
{
    int *ptr = num;
    int temp_max = *ptr;
    int *ptr2;
    for (int i = 0; i < size; i++)
    {
        if (temp_max < ptr[i + 1])
        {
            temp_max = ptr[i + 1];
        }
        else
        {
            continue;
        }
    }
    ptr2 = &temp_max;

    cout << "The maximum element in array is: " << *ptr2 << endl;
    return 0;
}

int l4q3()
{
    int num[] = {87, 32, 5, 92, 31, 54};
    int size = sizeof(num) / sizeof(int);
    findMax(num, size);

    return 0;
}

int main()
{
    startlab1();
    l4q3();
    return 0;
}
```

OUTPUT:



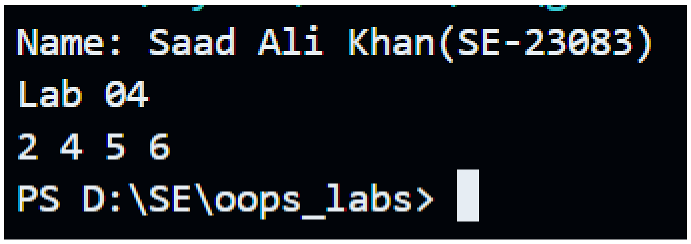
```
Name: Saad Ali Khan(SE-23083)
Lab 04
The maximum element in array is: 92
PS D:\SE\oops_labs>
```

4. Write a function removeDuplicates that takes an integer array and its size as arguments and removes duplicates, keeping only the first occurrence of each element.

CODE:

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

int startlab1()
{
    cout << "Name: Saad Ali Khan(SE-23083)" << endl;
    cout << "Lab 04" << endl;
    return 0;
}

int removeDuplicates(int arr[], int n)
{

    if (n == 0 || n == 1)
        return n;

    int temp[5];

    int j = 0;

    for (int i = 0; i < n - 1; i++)
        if (arr[i] != arr[i + 1])
            temp[j++] = arr[i];

    temp[j++] = arr[n - 1];

    // Modify original array
    for (int i = 0; i < j; i++)
        arr[i] = temp[i];

    return j;
}

int l4q4()
{
    int arr[] = {2, 2, 4, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    n = removeDuplicates(arr, n);

    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}

int main()
{
    startlab1();
    l4q4();
    return 0;
}
```

OUTPUT:

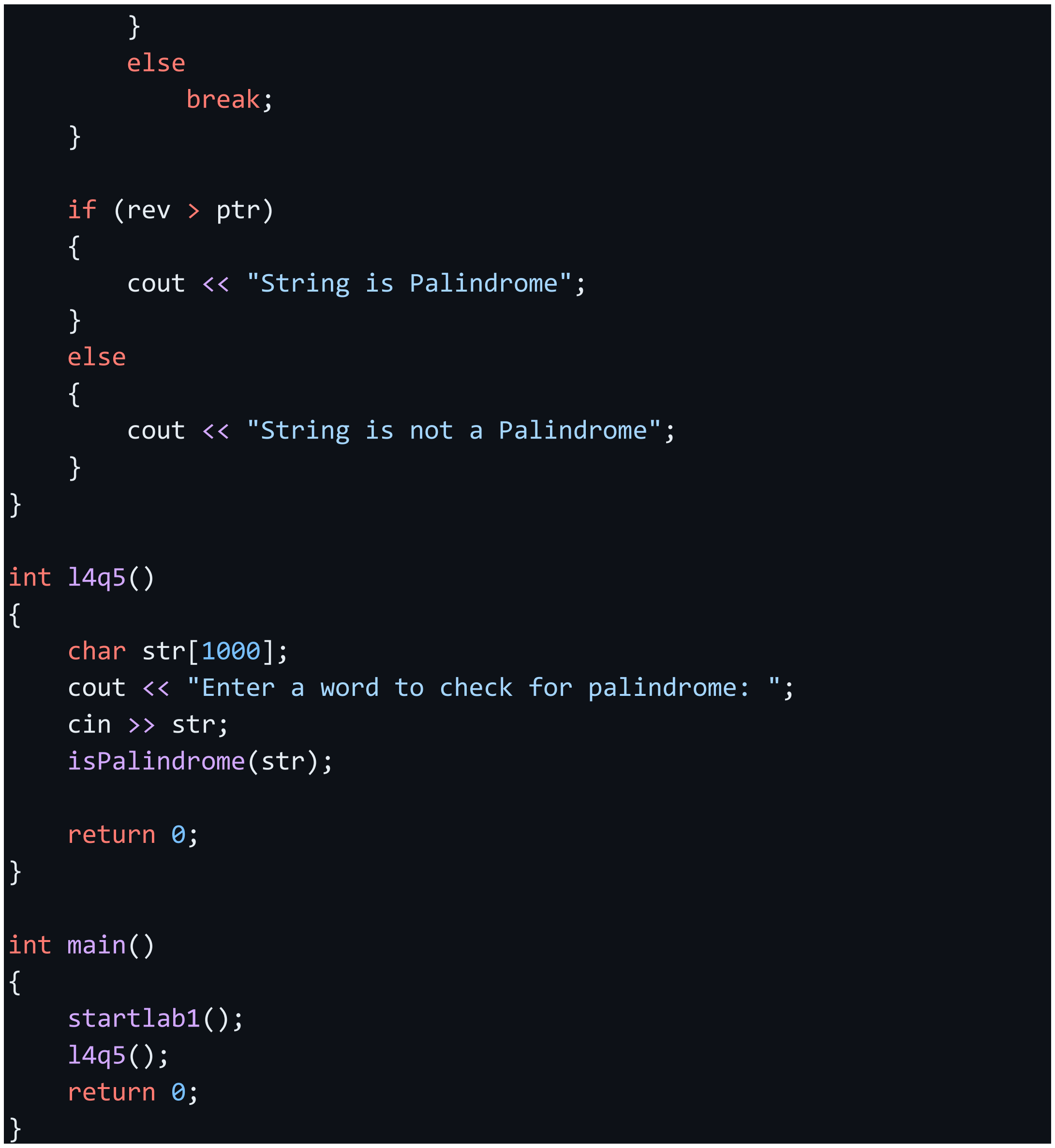


```
Name: Saad Ali Khan(SE-23083)
Lab 04
2 4 5 6
PS D:\SE\oops_labs>
```

5. Write a function isPalindrome that takes a string as an argument and returns true if the string is a palindrome, false otherwise. Use pointers to check for palindrome.

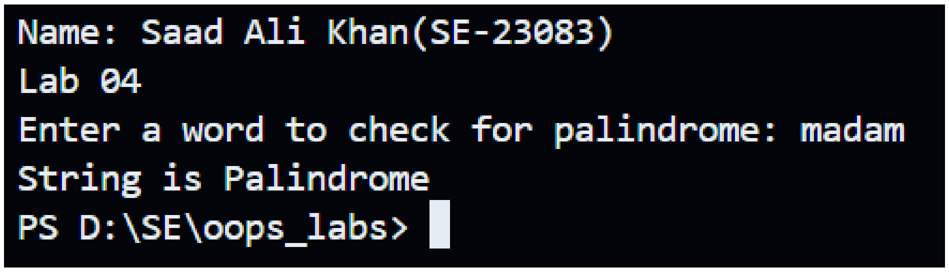
CODE:

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

int startlab1()
{
    cout << "Name: Saad Ali Khan(SE-23083)" << endl;
    cout << "Lab 04" << endl;
    return 0;
}

void isPalindrome(char *string)
{
    char *ptr, *rev;

    ptr = string;

    while (*ptr != '\0')
    {
        ++ptr;
    }
    --ptr;

    for (rev = string; ptr >= rev;)
    {
        if (*ptr == *rev)
        {
            --ptr;
            rev++;
```



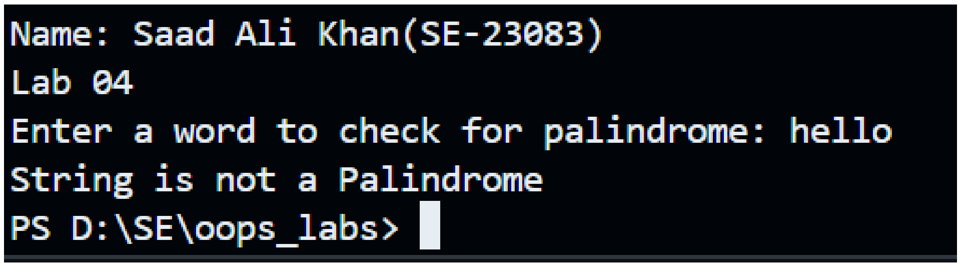
```
        }
        else
            break;
    }

    if (rev > ptr)
    {
        cout << "String is Palindrome";
    }
    else
    {
        cout << "String is not a Palindrome";
    }
}

int l4q5()
{
    char str[1000];
    cout << "Enter a word to check for palindrome: ";
    cin >> str;
    isPalindrome(str);

    return 0;
}

int main()
{
    startlab1();
    l4q5();
    return 0;
}
```

OUTPUT:
Test Case1:



```
Name: Saad Ali Khan(SE-23083)
Lab 04
Enter a word to check for palindrome: madam
String is Palindrome
PS D:\SE\oops_labs>
```

Test Case2:



```
Name: Saad Ali Khan(SE-23083)
Lab 04
Enter a word to check for palindrome: hello
String is not a Palindrome
PS D:\SE\oops_labs>
```