National University of Computer and Emerging Sciences

Object Oriented Programming (CS1004)

Date: Feb 27, 2024

Course Instructor(s)

Mr. Aamir Rahim

Ms. Anosha Khan

Ms. Arooj Khalil

Ms. Samin Iftikhar

Mr. Uzair Naqvi

Mr. Waqas Manzoor

Sessional-I Exam

Total Time: 1 Hour

Total Marks: 40

Total Questions: 02

Semester: SP-2024

Campus: Lahore

Dept: FAST School of Computing

Student Name	Roll No	Section	Student Signature

Vetted by	Vetter Signature

IMPORTANT INSTRUCTIONS: Answer in the space provided. **Answers written on rough sheet will not be marked**. Do not use pencil or red ink to answer the questions**.** In case of confusion or ambiguity make a reasonable assumption.

CLO # 4: Apply good programming practices

Q1: [4x5 = 20 marks] Short Questions

Part (a) Write output of the code segment below. (There is no syntax error in the code.)

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

void Swap(int*& a, int*& b)
{
	int* temp = a;
	a=b;
	b=temp;
}
```

```
void main()
{
	int a=5;
	int b=10;
	int* ptr1 = &a;
	int* ptr2 = &b;
	int** ptr3 = &ptr1;
	cout<<"Data = "<<**ptr3<<endl;
	int* temp1 = ptr1;
	int* temp2 = ptr2;
	Swap(temp1, temp2);
	cout<<"-------------"<<endl;
	cout<<"*ptr1 = "<<*ptr1<<endl;
	cout<<"*ptr2 = "<<*ptr2<<endl;
}
```

Output:

```
5
5
10
```

Part (b): Write output of the code segment below. If there is any error, clearly mention the error. (There is no syntax error in this code.)

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

int* SomeFunction()
{
	int abc = 50;
	return &abc;
}

void main()
{
	int* ptr1 = SomeFunction();
	cout<<"Data = ";
	cout<<*ptr1<<endl;
}
```

Output/Error:

Error(Dangling Pointer)

Part (c) Write the output of the code segment given below. (There is no syntax error in this code.)

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

void SomeFunction(int* arr, int size) {
    int* ptr1 = arr;
    int* ptr2 = arr + size - 1;
    while(ptr1 < ptr2) {
        *ptr1 = *ptr2;
        ptr1 = ptr1+2;
        ptr2--;
    }
}
```

```
int main() {
    int nums[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int* ptr = nums;
    SomeFunction(ptr, 10);
    for(int i = 0; i < 10; ++i) {
        cout << nums[i] << " ";
    }
    return 0;
}
```

Output:

```
10,2,9,4,8,6,7,8,9,10
```

Part (d) For the code segment given below, write output/error. In case of crash, highlight the line where program will crash. (There is no syntax error in this code.)

[THIS QUESTION IS NOT FOR BCS-2C]

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

int* GetData(int xyz)
{
	int* ptr = 0;
	if(xyz%2 == 0)
	{
		ptr = new int[5];
		for(int i=0; i<5; i++)
			ptr[i] = i+1;
	}
	return ptr;

}
```

```
int main() {
	int* array1[10];
	for(int i=0 ; i<10 ; i++)
	{
		array1[i] = GetData(i);
	}
	for(int i=0; i<10; i++)
	{
		for(int j=0; j<5 ; j++)
		{
			array1[i][j] = array1[i][j] *2;
			cout<<array1[i][j]<<" ";
		}
		cout<<endl;
	}
	//Assume we have Deallocation code here that
//successfully deallocates the memory.
}
```

Output/Error:

2,4,6,8,10
Null Exception

Part (d) ***[FOR BCS-2C ONLY]***

Consider the following program, **give C++ code for the class Point.** The distance formula is d = sqrt(dx*dx + dy*dy). The function sqrt is available in the C++ standard library.	**int main() {**
	Point p1(10,20);
	Point p2(30,50);
	cout << p1.distance(p2);
	return 0;
	}

Solution:

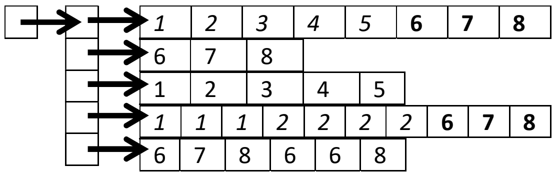
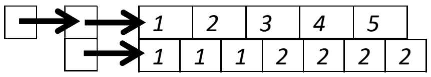
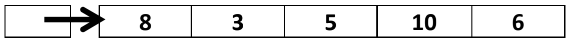
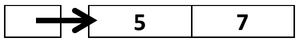
CLO # 3: Model an algorithmic solution for a given problem

Q2: [20 marks]

A program is getting multiple integer arrays (each array of variable size). It needs to keep only those arrays which end with a specific subArray. Your task is to write a function that takes a ListOfIntArrays (int**) and an ArrayToFind (int*) i.e. SubArray. The function should remove all the arrays (from ListOfIntArrays) that do not end with ArrayToFind. Prototype of the function is given below:

void FilterData(int& ListOfIntArrays, int*& LenghtsOfArrays, int*& ArrayToFind, int& SizeOfArrayToFind, int& TotalIntArrays)**

Sample run below shows the values of required variables and arrays' content before and after the function call for **ArrayToFind = {6,7,8}** and **SizeOfArrayToFind = 3**.

Before Function Call	After Function Call	Explanation
ListOfIntArrays:  1 2 3 4 5 6 7 8 6 7 8 1 2 3 4 5 1 1 1 2 2 2 2 6 7 8 6 7 8 6 6 8	**ListOfIntArrays:** 	All the arrays that do not end with ArrayToFind = {6,7,8} have been removed. The array that ends with {6,7,8} but does not have any other data has also been removed.
TotalIntArrays: 5	**TotalIntArrays:** 2	Total no. of int arrays in ListOfIntArrays
LenghtsOfArrays: 	**LenghtsOfArrays:** 	Array Containing Lengths of all 1D int arrays in ListOfIntArrays.

Functionality Explanation:
Row 1, {1,2,3,4,5,**6,7,8**}: *Not Removed*, as ArrayToFind {6,7,8} found at the end.
Row 2, {**6,7,8**}: *Removed*, as ArrayToFind {6,7,8} found at end but there wasn't any other data in this array.
Row 3, {1,2,3,4,5}: *Removed*, as ArrayToFind {6,7,8} NOT Found at the end.
Row 4, {1,1,1,2,2,2,2,**6,7,8**}: *Not Removed*, as ArrayToFind {6,7,8} found at the end.
Row 5, {6,7,8,6,6,8}: *Removed*, as ArrayToFind {6,7,8} NOT Found at the end.
Note that the data of ArrayToFind {6,7,8} has also been removed from original data arrays (ListOfIntArrays).

Make sure that arrays do not consume extra space. Also there should not be any memory leakage or dangling pointer.

void FilterData(int& ListOfIntArrays, int*& LenghtsOfArrays, int*& ArrayToFind, int& SizeOfArrayToFind, int& TotalIntArrays)**
{
//Start your code here…

```
// Function to filter arrays based on whether they end with a specified subarray
void FilterData(int**& arr, int*& arrLenghts, int& totalArrays, int* subArr, int
sizeOfSubArray)
{
    int required_arrays = 0;  // Count of arrays that meet the condition
    int** result1 = new int* [totalArrays];  // Array to store filtered arrays

    // Iterate through each array in the input array of arrays
    for (int i = 0; i < totalArrays; i++)
    {
        // Check if the array ends with the specified subarray
        if (EndsWithSubArray(arr[i], subArr, arrLenghts[i], sizeOfSubArray))
        {
            required_arrays++;  // Increment the count of arrays that meet the condition

            // Create a new array without the ending subarray
            int new_size = arrLenghts[i] - sizeOfSubArray;
            result1[i] = new int[new_size];

            // Copy elements from the original array to the new array
            for (int j = 0; j < new_size; j++)
            {
                result1[i][j] = arr[i][j];
            }

            arrLenghts[i] = new_size;  // Update the length of the original array
            delete[] arr[i];  // Deallocate memory for the original array
        }
        else
        {
            delete[] arr[i];  // Deallocate memory for arrays that don't meet the condition
            result1[i] = 0;  // Set corresponding entry in result1 to null
        }
    }

    delete[] arr;  // Deallocate memory for the original array of arrays
    arr = new int* [required_arrays];  // Create a new array of arrays to store filtered
arrays
    int* temp_arr_lengths = new int[required_arrays];  // Temporary array to store updated
array lengths
    int j = 0;  // Index for the new array of arrays

    // Iterate through the result1 array to update arr and arrLenghts
    for (int i = 0; i < totalArrays; i++)
    {
        if (result1[i])
        {
            arr[j] = result1[i];  // Assign the filtered array to arr
            temp_arr_lengths[j] = arrLenghts[i];  // Assign the updated array length to
temp_arr_lengths
            j++;  // Move to the next index in the new array of arrays
        }
    }

    delete[] arrLenghts;  // Deallocate memory for the original array of array lengths
    arrLenghts = temp_arr_lengths;  // Update arrLenghts with the updated array lengths
    totalArrays = required_arrays;  // Update the total number of arrays
}



// Function to check if an array ends with a specified subarray
bool EndsWithSubArray(int* arr, int* subArray, int size, int sizeOfSubArray)
{
    int j = size - 1;  // Start from the end of the array
    int i = sizeOfSubArray - 1;  // Start from the end of the subarray

    // Iterate through the subarray and array from the end
    for (; i >= 0; i--)
    {
        // Check if the elements don't match
        if (arr[j] != subArray[i])
            return false;
        j--;  // Move to the previous element in the array
    }

    // Check if the entire array has been iterated (no remaining elements)
    if (j == -1)
        return false;

    // If there are remaining elements in the array, return true
    return true;
}
```