Name: Deepanshi
Student #: 456807 Email ID: d.807@mybvc.ca

SODV 1101
Programming Fundamentals
Winter 2024
Assignment 3
Instructor: Mahbub Murshed Due Date: 2024-04-19

Directions:

- Written questions can be submitted via soft copy or a scanned document and submitted by d2l dropbox, **as a single doc file or single pdf file**. Answers should be written clearly and coherently. **Programming questions must be submitted in repl.it.**
- Assignments must be submitted by the posted due date, or be subject to the course's late policy. Work must be completed individually and in accordance with the academic integrity policy.
- All codes shown in this assignment are compiled and tested except for questions that aim to find errors in a C++ program.
- For all written questions, you may assume that the usual header files are included as follows:

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

Note: The assignment's written questions not submitted as a single file will not be graded!

Written Questions	
1	/ 10
2	/ 10
3	/ 10
Programming Questions	
Quality	/ 20
Functionality	/ 50
Total	
/ 100	

Assignment 3: Written Questions

1. Give concise answers to the following questions / 10

a) What are Null Pointers? Can a pointer store the address of cells of an array? Give examples. /1+2

ANS: A **null value** (often shortened to null) is a special value that means something has no value. When a pointer is holding a null value, it means the pointer is not pointing at anything. Such a pointer is called a null pointer.

Yes, a pointer can store the address of cells of an array. In fact, arrays are often accessed using pointers in languages like C and C++

Example: (For null pointers)

(I)
```
#include <iostream>

using namespace std;
int main () {
   int  *ptr = NULL;
   cout << "The value of ptr is " << ptr ;

   return 0;
}
```
OUTPUT: The value of ptr is 0

(ii)
```
#include <iostream>
using namespace std;
int main() {
    int arr[] = {10, 20, 30, 40, 50};
     int *ptr = arr; // Pointer storing the address of the first element of
the array

    // Accessing array elements using pointer arithmetic
    cout << "Array elements: ";
    for (int i = 0; i < 5; ++i) {
        cout << *(ptr + i) << " "; // Dereferencing the pointer to access
array elements
    }
return 0;
}
```

Yes, a pointer stores the address of cells of an array. The pointer **ptr** is initialized with the address of the first element of the array arr. Then, array elements are accessed using pointer arithmetic (*(ptr + i)), which allows traversing the array by adding an offset to the pointer.

OUTPUT: Array elements: 10 20 30 40 50

b) What are the address-of and the dereference operators, and what do they do? /2+2

ANS: The **address-of operator** (&) is used to obtain the memory address of a variable. It returns a pointer to the memory location where the variable is stored.

Example:

```
#include <iostream>
using namespace std;
int main() {
   int x = 10;
   cout << "Address of x: " << &x <<endl;
   float a;
   cout << "Address of a: " << &a <<endl;
   return 0;
}
```

OUTPUT: Address of x: 0x7ffe650eefdc

```
Address of a: 0x7ffe650eefd8
```

In this example, &x returns the memory address of the variable x. The output will be something like 0x7ffe650eefdc, which is the hexadecimal representation of the memory address where x is stored.

You can also use the address-of operator with other types of variables, not just integers like, in the example when we declared the variable in float data type and &a returns the memory address of the variable a. The output will be like this 0x7ffe650eefd8.

Remember, when you use the address-of operator, you're dealing with pointers, as the result of &x is of pointer type (int* in this case), not the value itself.

The **dereference operator** (*) is used to access the value stored at a memory address pointed to by a pointer.

Example:

```
#include <iostream>
using namespace std;
int main() {
   int x = 10;
   int *ptr = &x; // ptr now holds the memory address of x

     cout << "Value of x: " << *ptr <<endl; // dereferencing ptr to get the
value of x
   return 0;
```

}

OUTPUT: Value of x: 10

In this example, *ptr dereferences the pointer ptr to access the value stored at the memory address it points to, which is the value of x. So, *ptr essentially gives you the value of x, which is 10.

Dereferencing is a fundamental concept when working with pointers because it allows you to access and manipulate the data that the pointer is pointing to.

c) What is the difference between pass-by-value and pass-by-reference? / 3

ANS :

S.No	Pass by Value	Pass by Reference
1.	Passes an argument by value.	Passes an argument by reference.
2.	In pass-by-value, the actual value that is passed as an argument is not changed after performing some operation on it.	In Pass by reference the actual value that is passed as an argument is changed after performing some operation on it.
3.	When pass-by value is used, it creates a copy of that variable into the stack section in memory.	When a pass-by reference is used, it creates a copy of the reference of that variable into the stack section in memory.
4.	This method does not use pointers.	This method uses the pointers.

5.	**EXAMPLE:**
	#include<iostream>
	using namespace std;
	
	void my_function(int x) {
	  x = 50; // Modifying the local copy of x
	  cout << "Value of x from my_function: " << x << endl;
	}
	int main() {
	  int x = 10;
	  my_function(x); // Passing the value of x to my_function
	  cout << "Value of x from main function: " << x;
		EXAMPLE:
		#include<iostream>
		using namespace std;
		
		void my_function(int &x) {
		  x = 50;
		    cout << "Value of x from my_function: " << x << endl;
		}
		
		int main() {
		  int x = 10;
		  my_function(x);
		   cout << "Value of x from main function: " << x;
		  return 0;
		}

	return 0; } **OUTPUT**: Value of x from my_function: 50 Value of x from main function: 10	**OUTPUT**: Value of x from my_function: 50 Value of x from main function: 50

2. / 10

a) What are the values of the variables a, b, and c after the function call q2a(a, b, c) completes in the program below? / 5

```
void q2a(int& x, int& y, int z)
{
    for (; x > 0; --x)
    {
        if (y < z)
            y = z + x;
        else
            z = y + x;
    }
}

int main()
{
    int a = 10, b = 5, c = 3;
    q2a(a, b, c);

    //a=? b=? c=?

    return 0;
}
```

Ans: Firstly, according to this code, we do not give the print statement (cout) so the value can be seen on a console. For finding the value of a,b,c after giving the cout<<"The value of a = "<<a<<endl;
cout<<"The value of b = "<<b<<endl;
cout<<"The value of c = "<<c<<endl;
we can find the a,b,c.

OUTPUT:

```
The value of a = 0
The value of b = 60
The value of c = 3
```

COMPLETE CODE:

```
#include<iostream>
using namespace std;
void q2a(int& x, int& y, int z)
{
    for (; x > 0; --x)
    {
        if (y < z)
            y = z + x;
        else
            z = y + x;
    }
}
int main()
{
    int a = 10, b = 5, c = 3;
    q2a(a, b, c);
   cout<<"The value of a = "<<a<<endl;
    cout<<"The value of b = "<<b<<endl;
    cout<<"The value of c = "<<c<<endl;
    return 0;
}
```

OUTPUT:

```
The value of a = 0
The value of b = 60
The value of c = 3
```

Explanation:

(I) When x = 10, 5<3 means (z is greater than y)that is false so the else statement comes into action where **z = y + x i.e z = 5+ 10 = 15.**

(ii) Now, when x = 9, y = 5, and z = 15, so, 5 < 15 means (y is smaller than z) that is true so the if statement comes into action where **y = z + x i.e y = 15 + 9 = 24**.

(iii) When x = 8, y = 24, and z = 15, 24<15 means (z is greater than y)that is false so the else statement comes into action where **z = y + x i.e z = 24 + 8 = 32.**

(iv) When x = 7, y = 24, and z = 32, so, 24 < 32 means (y is smaller than z) that is true so the if statement comes into action where **y = z + x i.e. y = 32 + 7 = 39.**

(v) When x = 6, y = 39, and z = 32, 39<32 means (z is greater than y)that is false so the else statement comes into action where **z = y + x i.e z = 39 + 6 = 45.**

(vi) When x = 5, y = 39, and z = 45, so, 39 < 45 means (y is smaller than z) that is true so the if statement comes into action where **y = z + x i.e. y = 45 + 5 = 50.**
(vii) When x = 4, y = 50, and z = 45, 50<45 means (z is greater than y)that is false so the else statement comes into action where **z = y + x i.e z = 50 + 4 = 54.**
(viii) When x = 3, y = 50 and z = 54, so, 50 < 54 means (y is smaller than z) that is true so the if statement comes into action where **y = z + x i.e. y = 54 + 3 = 57**.
(ix) When x = 2, y = 57, and z = 54, 57<54 means (z is greater than y) that is false so the else statement comes into action where **z = y + x i.e. z = 57 + 2 = 59.**
(x) When x = 1, y = 57, and z = 59, 57<59 means (y is smaller than z) that is true so the if statement comes into action where **y = z + x i.e. y = 59 + 1 = 60**.
(xi) When x = 0 , the cursor move out from the loop and x become 0 and y = 60 and z = 59.

```
 void q2a(int& x, int& y, int z)
```

When we call the function in main q2a(a,b,c). So a becomes 0 as the value is passed by reference it will make changes the same with b is called by reference so b becomes 60 and z is not passed by reference it will remain unchanged so initially z = 3 so after calling the value in the main function it becomes 3 (no change).

b) What is the exact output of the following program? / 5

```
string q2b(char* l, int* c, int cSize)
{
    string o = "";
    for (int i = 0; i < cSize; ++i)
    {
        if (c[i] < 0)
            continue;
        o += l[c[i]];
    }
    return o;
}

int main()
{
    char l[] = {'o','a','c','l','e','r','t','i','u','n','h','g','d'};
    int c[] = { 2, 0, 12, -1, 7, 9, 11 };
    cout << q2b(l, c, 7) << endl;
    return 0;
}
```

ANS:

Output: coding

```
#include<iostream>
using namespace std;
string q2b(char* l, int* c, int cSize)
{
    string o = "";
    for (int i = 0; i < cSize; ++i)
    {
        if (c[i] < 0)
            continue;
        o += l[c[i]];
    }
    return o;
}

int main()
{
    char l[] = {'o','a','c','l','e','r','t','i','u','n','h','g','d'};
    int c[] = { 2, 0, 12, -1, 7, 9, 11 };
    cout << q2b(l, c, 7) << endl;
    return 0;
}
```

Explanation:

Inside the function:

- It initializes an empty string o.
- It iterates over the integer array c from index 0 to cSize - 1.
- For each valid index i, if the value of c[i] is less than 0, it skips to the next iteration using continue.
- Otherwise, it appends the character at index c[i] from the character array l to the output string o.

main **function**:

- It initializes two arrays:
  - char l[]: An array of characters representing letters.
  - int c[]: An array of integers representing indices.
- It calls the q2b function with the arrays l and c, and cSize as 7 (the size of the c array).
- It prints the returned string from q2b function to the console.

Explanation of the main **function execution**:

- The char array l contains the letters 'o', 'a', 'c', 'l', 'e', 'r', 't', 'i', 'u', 'n', 'h', 'g', 'd'.
- The int array c contains the indices: 2, 0, 12, -1, 7, 9, 11.
- When the q2b function is called with cSize as 7, it will only consider the first 7 elements of array c.
- It will skip the index with value -1.
- So, it will concatenate the characters at indices 2, 0, 12, 7, 9, 11 from array l, which are 'c', 'o', 'd', 'i', 'n', 'g'.
- Thus, the output of the program will be "coding".

3. The goal of the code below is to read a list supplied by the user and print it back in reverse order. Identify all errors (logical and syntax), specify them, and fix them. / 10

```
#include <iostream>
using namespace std;
int main()
{
    string repeat = "Y";
    while (repeat == "Y");
    {
        int count;
        cout << "How many numbers are on your list? ";
        cin >> count;
        double data[] = new double[count];
        cout << "Enter the numbers: " << endl;
        while (int i = 0; i < count; ++i)
            cin >> data[i];
        cout << "List in reverse order: " << endl;
        for (int i = 0; i < count; ++i)
            cout << data[count - i - 1] << endl;
        do
        {
            cout << "Try another list? [Y/N] ";
            cin >> repeat;
        } while (repeat != "Y" && repeat != "N")
    }
    return 0;
}
```

After the fix, the following will be a sample input/output from the above code:

```
How many numbers are on your list? 3
Enter the numbers:
45
67
12
List in reverse order:
12
67
45
Try another list? [Y/N]Y
How many numbers are on your list? 4
```

```
Enter the numbers:
1
2
3
4
List in reverse order:
4
3
2
1
Try another list? [Y/N]N
Process returned 0 (0x0)   execution time : 22.005 s
```

ANS:

1) **Syntax Error**: double data[] = new double[count]; we should use new to dynamically allocate memory. double *data = new double[count];

2) Syntax Error: while (int i = 0; i < count; ++i); The while loop is wrongly declared. It should be declared using the for loop construct, not while.
for(int i = 0; i < count; ++i)

3) Syntax Error: while (repeat != "Y" && repeat != "N") missing of semicolon
after do-while loop declaration.

4) Logic Error: while (repeat == "Y"); Remove the semicolon to ensure further statements are executed.

Also, delete[] data; // Deallocate the created memory allocation is considered a logical error

CORRECTED CODE:

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

int main() {
   string repeat = "Y";
   while (repeat == "Y") { // remove semicolon to ensure further statements are executed
(logic error)
      int count;
      cout << "How many numbers are on your list? ";
      cin >> count;
      double *data = new double[count]; // Corrected memory allocation
      cout << "Enter the numbers: " << endl;
      for (int i = 0; i < count; ++i){ // Corrected loop
         cin >> data[i];
```

```
        }
        cout << "List in reverse order: " << endl;
        for (int i = 0; i < count; ++i){
            cout << data[count - i - 1] << endl;
        }
        delete[] data; // Deallocate (logic)
        do {
            cout << "Try another list? [Y/N] ";
            cin >> repeat;
        } while (repeat != "Y" && repeat != "N"); // Corrected semicolon
    }
    return 0;
}
```

References

1. Alex, January 10, 2024. Null Pointer. Learn C++, from https://www.learncpp.com/cpp-tutorial/null-pointers/

2. TutorialPoints. Difference Between pass by value and pass by reference in C++, from https://www.tutorialspoint.com/differences-between-pass-by-value-and-pass-by-reference-in-cplusplus

3. Educative. Pass by value vs. pass by reference, from https://www.educative.io/answers/pass-by-value-vs-pass-by-reference

Program 1: Complex Numbers (Please submit in repl.it)

Learning Objective
- Use Structures to group related but dissimilar data types.
- Summarize how memory is addressed inside a computer.
- Decide where to use the “pass-by-value” or the “pass-by-reference” technique based on application purpose and the programming need.
- Demonstrate logical thinking by using programming syntax and strategies.

Declare a structure to represent a complex number (a number having a real part and an imaginary part). Write C++ functions to add, subtract, and multiply two complex numbers.

Hints:
Addition: (a+b**i**) + (c+d**i**) = (a+c) + (b+d)**i**
Subtraction: (a+b**i**) - (c+d**i**) = (a-c) + (b-d)**i**
Multiplication: (a+b**i**) * (c+d**i**) = (ac − bd) + (ad + bc)**i**

To know more about complex numbers, please visit: https://www.mathsisfun.com/numbers/complex-numbers.html

Sample input/output:

```
Please enter the first complex number:
Enter real and imaginary parts only:
(Please separate inputs by space or the enter key and do not use 'i' after the complex part)
4 -8

Please enter the second complex number:
Enter real and imaginary parts only:
(Please separate inputs by space or the enter key and do not use 'i' after the complex part)
2 -4
Addition = 6-12i
subtraction = 2-4i
multiplication = -24-32i

Process returned 0 (0x0)   execution time : 14.684 s
Press any key to continue.
```

Program 2: Student Rank List   (Please submit in repl.it)

Learning Objective

- Use Structures to group related but dissimilar data types.
- Summarize how memory is addressed inside a computer.
- Decide where to use the “pass-by-value” or the “pass-by-reference” technique based on application purpose and the programming need.
- Demonstrate logical thinking by using programming syntax and strategies.

Create a structure to store details of ‘n’ students (id, name, marks in five subjects). Randomly populate all student’s records (use random id in range (100 to 999), random name, random marks in range (20.. 100)). Create a function to display a list of students based on their average marks (descending). Finally, display students who have failed in more than one subject (consider 40 as the pass mark).

Sample Input/Output:

```
Please enter the total students:
5
Randomly populated students:
Student Id: 737, Name: CYEAJXOT, Average Marks: 70.2
Subjects marks: 94, 95, 22, 41, 99,
Student Id: 551, Name: NSBRQ, Average Marks: 67.6
Subjects marks: 94, 82, 68, 56, 38,
Student Id: 432, Name: BQEU, Average Marks: 49.6
Subjects marks: 54, 99, 32, 21, 42,
Student Id: 450, Name: HLLQZI, Average Marks: 57.4
Subjects marks: 88, 29, 74, 76, 20,
Student Id: 175, Name: MAKUT, Average Marks: 49
Subjects marks: 43, 51, 74, 55, 22,

List of sorted students according to their average marks:
Student Id: 737, Name: CYEAJXOT, Average Marks: 70.2
Subjects marks: 94, 95, 22, 41, 99,
Student Id: 551, Name: NSBRQ, Average Marks: 67.6
Subjects marks: 94, 82, 68, 56, 38,
Student Id: 450, Name: HLLQZI, Average Marks: 57.4
Subjects marks: 88, 29, 74, 76, 20,
Student Id: 432, Name: BQEU, Average Marks: 49.6
Subjects marks: 54, 99, 32, 21, 42,
Student Id: 175, Name: MAKUT, Average Marks: 49
Subjects marks: 43, 51, 74, 55, 22,

List of students who failed in more than one subject:
Student Id: 450, Name: HLLQZI, Average Marks: 57.4
Subjects marks: 88, 29, 74, 76, 20,
Student Id: 432, Name: BQEU, Average Marks: 49.6
Subjects marks: 54, 99, 32, 21, 42,
Process returned 0 (0x0)   execution time : 0.831 s
Press any key to continue.
```

Evaluation

This assignment has 2 main components, assigned marks as follows:

Task	**Marks**
Written Questions	30
Code Quality and Functionality	70
Total	100

Marks for written questions are indicated beside each question.
The **following rubrics will be followed for assessing the program code:**

Excellent (100%)	***Competent (80%)***	***Satisfactory (50%)***	***Unsatisfactory (0%)***
● *Code contains no compile-time errors.* ● *Code contains no logical errors.* ● *Code produces the exact correct output and does not crash for valid inputs.* ● *Code is well organized, clear, and easy to read.* ● *Code is consistent throughout and comments are used where needed.*	● *Code contains no compile-time errors.* ● *Code produces the exact correct output except for a few formatting issues / incorrect data type issues.* ● *Code does not crash for valid inputs.* ● *Code produces the correct output most of the time. However, for some valid inputs, it shows the incorrect output.* ● *Minor logical errors are present.*	● *Code contains no compile-time errors.* ● *Code is disorganized or could use some refactoring.* ● *Code contains major logical errors and does not match the expected output.*	● *Code contains compile-time errors.* ● *Code contains major logical errors.* ● *Code is very difficult to read.* ● *Code does not execute.*