FEU INSTITUTE OF TECHNOLOGY

COLLEGE OF COMPUTER STUDIES AND MULTIMEDIA ARTS

CCS0015L (DATA STRUCTURES AND ALGORITHMS)

EXERCISE 6

RECURSION

Student Name / Group Name:	Fernando Dane I. Borja
Members (if Group):	**Name** / **Role** (no entries)
Section:	TC05 BSIT – CST
Professor:	Joseph Calleja

Name	**Role**

I. PROGRAM OUTCOME/S (PO) ADDRESSED BY THE LABORATORY EXERCISE

- Identify, analyze and solve computing problems using fundamental principles of mathematics and computing sciences. [PO: B]

II. COURSE LEARNING OUTCOME/S (CLO) ADDRESSED BY THE LABORATORY EXERCISE

- Apply the fundamental principles of data structures and algorithms: concepts of abstract data; types of common data structures used; description, properties, and storage allocation of data structures. [CLO: 2]

III. INTENDED LEARNING OUTCOME/S (ILO) OF THE LABORATORY EXERCISE

At the end of this exercise, students must be able to:

- Learn how to create a base case and inductive step.
- Implement recursion in solving programming problems that require it.

IV. BACKGROUND INFORMATION

What is Recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), In-order/Preorder/Post-order Tree Traversals, DFS of Graph, etc.

What is base condition in recursion?

In recursive program, the solution to base case is provided and solution of bigger problem is expressed in terms of smaller problems.

```
int fact(int n)
{
    if (n < = 1) // base case
        return 1;
    else
        return n*fact(n-1);
}
```

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

Need of Recursion

Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write. It has certain advantages over the iteration technique which will be discussed later. A task that can be defined with its similar subtask, recursion is one of the best solutions for it. For example, The Factorial of a number.

V. LABORATORY ACTIVITY

ACTIVITY 6.1: Duplicates

Write a C++ program to find the frequency of each element in a sorted array containing duplicates using recursion.

Sample Output:
If the array given is: int arr[] = {1, 2, 2, 3, 4, 4, 4, 5, 5};

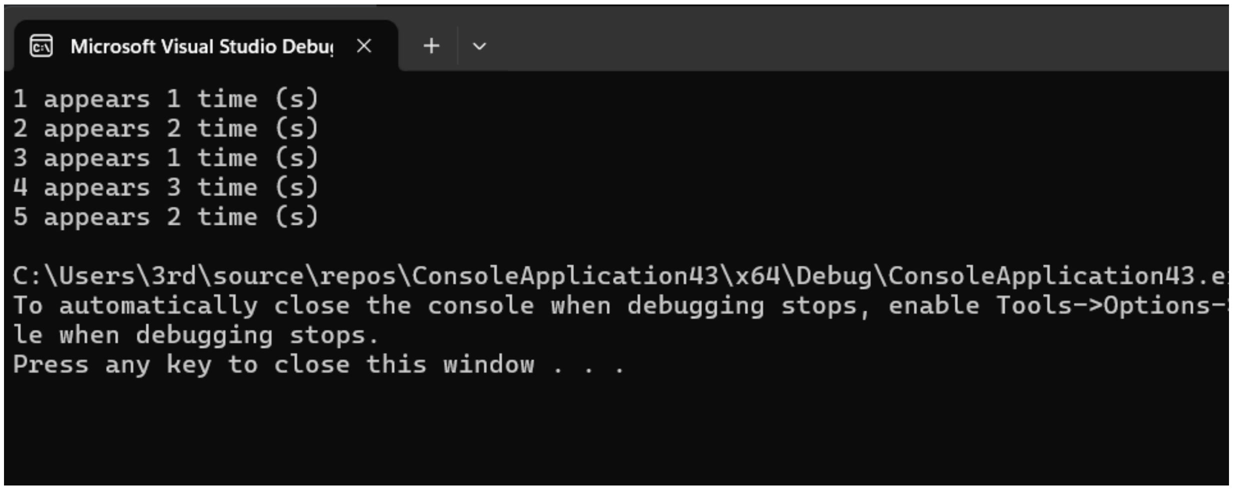
```
1 appears 1 time(s)
2 appears 2 time(s)
3 appears 1 time(s)
4 appears 3 time(s)
5 appears 2 time(s)
```

Program: (save as [surname_6_1.cpp])

Note: Write your complete working program here.

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

void findFrequency(int arr[], int n, int index) {
    if (index == n) {
        return;
    }

    int count = 1;
    while (index < n - 1 && arr[index] == arr[index + 1]) {
        count++;
        index++;
    }

    cout << arr[index] << " appears " << count << " time (s)" << endl;

    findFrequency(arr, n, index + 1);
}

int main() {
    int arr[] = { 1, 2, 2, 3, 4, 4, 4, 5, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);

    findFrequency(arr, n, 0);

    return 0;
                                            }
```

Output:(screenshot of the output)



```
1 appears 1 time (s)
2 appears 2 time (s)
3 appears 1 time (s)
4 appears 3 time (s)
5 appears 2 time (s)

C:\Users\3rd\source\repos\ConsoleApplication43\x64\Debug\ConsoleApplication43.e
To automatically close the console when debugging stops, enable Tools->Options-
le when debugging stops.
Press any key to close this window . . .
```

ACTIVITY 6.2: Tower of Hanoi

Write a program that will emulate the game Tower of Hanoi. The Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1) Only one disk can be moved at a time.
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3) No disk may be placed on top of a smaller disk.

The program will ask the user how many rods will be used (either 2 or 3, if the user inputs a different number, the program will ask for another input), then it will display the different moves to solve the Tower of Hanoi. NOTE: The program must use a recursive function, not a looping structure.

Sample Output:

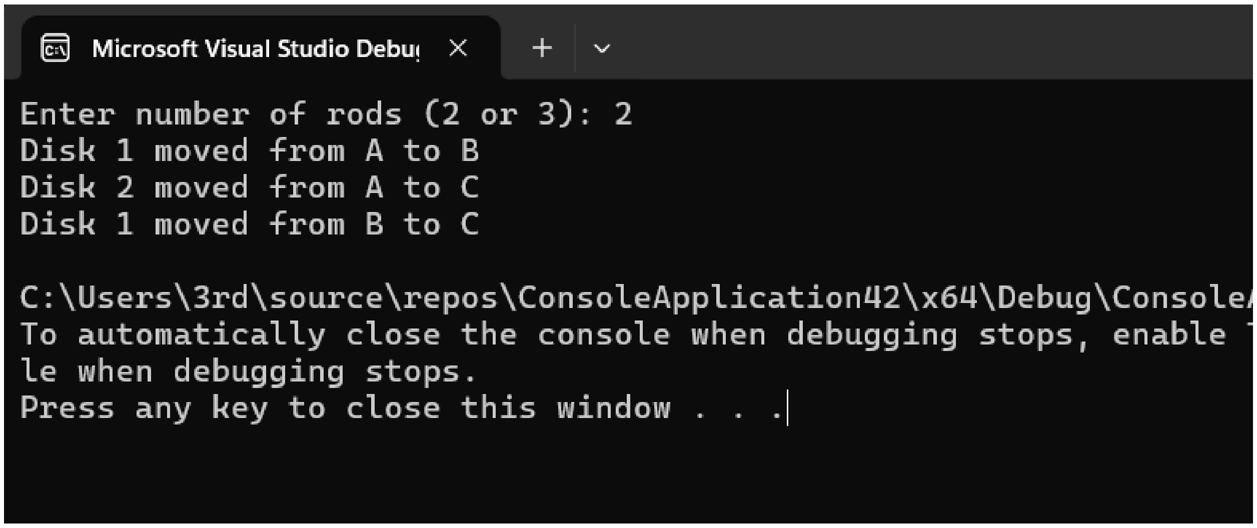
```
Input : 2
Output : Disk 1 moved from A to B
         Disk 2 moved from A to C
         Disk 1 moved from B to C

Input : 3
Output : Disk 1 moved from A to C
         Disk 2 moved from A to B
         Disk 1 moved from C to B
         Disk 3 moved from A to C
         Disk 1 moved from B to A
         Disk 2 moved from B to C
         Disk 1 moved from A to C
```

Program: (save as [surname_6_1.cpp])

Note: Write your complete working program here.

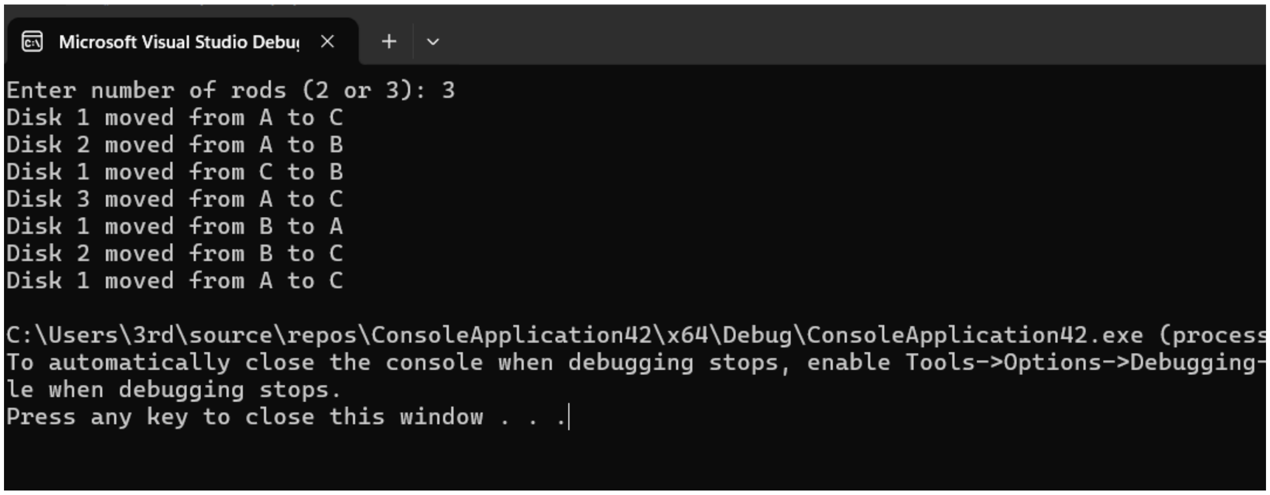
```
#include <iostream>
using namespace std;

void towerOfHanoi(int n, char fromRod, char toRod, char auxRod) {
    if (n == 1) {
        cout << "Disk 1 moved from " << fromRod << " to " << toRod << endl;
        return;
    }
    towerOfHanoi(n - 1, fromRod, auxRod, toRod);
   cout << "Disk " << n << " moved from " << fromRod << " to " << toRod <<endl;
    towerOfHanoi(n - 1, auxRod, toRod, fromRod);
}

void towerOfHanoiWrapper(int n, char fromRod, char toRod, char auxRod) {
    towerOfHanoi(n, fromRod, toRod, auxRod);
}

int main() {
    int numRods;
    char fromRod = 'A', toRod = 'C', auxRod = 'B';

    do {
       cout << "Enter number of rods (2 or 3): ";
       cin >> numRods;
    } while (numRods != 2 && numRods != 3);

    towerOfHanoiWrapper(numRods, fromRod, toRod, auxRod);

    return 0;
                                          }
```

Output:(screenshot of the output)



Microsoft Visual Studio Debu

```
Enter number of rods (2 or 3): 2
Disk 1 moved from A to B
Disk 2 moved from A to C
Disk 1 moved from B to C

C:\Users\3rd\source\repos\ConsoleApplication42\x64\Debug\Console
To automatically close the console when debugging stops, enable 
le when debugging stops.
Press any key to close this window . . .
```



Microsoft Visual Studio Debu

```
Enter number of rods (2 or 3): 3
Disk 1 moved from A to C
Disk 2 moved from A to B
Disk 1 moved from C to B
Disk 3 moved from A to C
Disk 1 moved from B to A
Disk 2 moved from B to C
Disk 1 moved from A to C

C:\Users\3rd\source\repos\ConsoleApplication42\x64\Debug\ConsoleApplication42.exe (process
To automatically close the console when debugging stops, enable Tools->Options->Debugging-
le when debugging stops.
Press any key to close this window . . .
```

VI. QUESTION AND ANSWER

Directions: Briefly answer the questions below.

• Explain the advantages and disadvantages of using recursion. The code can be made shorter with the aid of recursion. Compared to non-recursive functions, recursive functions are a little slower. It offers a clear and simple approach to writing code. Compared to the iterative programs, its space requirements are greater.
• Discuss how base case is identified in a given problem when implementing recursion. In the simplest scenario, we compute the outcome as soon as the function call's inputs are provided. Using one or more recursive calls to the same function, we compute the output in a recursive step. This brings the inputs closer to a base case by reducing their size or complexity.

VII. REFERENCES

- Wittenberg, Lee.(2018). Data structures and algorithms in C++. s.l.: Mercury Learning
- Baka, Benjamin(2017). Python data structures and algorithms : improve the performance and speed of your applications. Birmingham, U.K : Packt Publishing
- Downey, Allen.(2017). Think data structures : algorithms and information retrieval in Java. Sebastopol, CA: O'Reilly
- Chang, Kung-Hua(2017). Data Structures Practice Problems for C++ Beginners. S.I : Simple and Example
- Hemant Jain(2017). Problem Solving in Data Structures & Algorithms Using C++: Programming Interview Guide. USA: CreateSpace Independent Publishing Platform

RUBRIC:

Criteria	4	3	2	1	Score
Solution(x5)	A completed solution runs without errors. It meets all the specifications and works for all test data.	A completed solution is tested and runs but does not meet all the specifications nd/or work for all test data.	A completed solution is implemented on the required platform, and uses the compiler specified. It runs, but has logical errors.	An incomplete solution is implemented on the required platform. It does not compile and/or run.	

Program Design(x3)	The program design uses appropriate structures. The overall program design is appropriate.	The program design generally uses appropriate structures. Program elements exhibit good design.	Not all of the selected structures are appropriate. Some of the program elements are appropriately designed.	Few of the selected structures are appropriate. Program elements are not well designed.	
Completeness of Document(x2)	All required parts of the document are complete and correct(code, output of screenshots)	All required parts in the document are present and correct but not complete.	There are few parts of the document are missing but the rest are complete and correct.	Most of the parts of the document are missing and incorrect.	
					Total