Lab No 01

**OBJECT**

***To Study the difference in pseudo-code and algorithm and its implementation.***

***\* Problem Solving Skills***

***\* Pseudo-code***

***\* Algorithm***

***\* Implementation of pseudo-code & algorithm***

**THEORY**

A typical Programming task can be divided into two phases:

1. **Problem Solving Phase:**

* Produces an ordered sequence of number of steps that describe solution of problem.
* This sequence of steps is called **algorithm.**

1. **Implementation Phase:**

* Implement the program in some programming language.

**Steps In Problem Solving:**

1. First produce a general algorithm (one can use **pseudo-code**).
2. Refine the algorithm successively to get step by step detailed **algorithm** that is very close to a computer language.

**Pseudo-code:**

Pseudo-code is an artificial and informal language that helps programmers develops algorithms. Pseudo-code is very similar to everyday English.

**Algorithm:**

An algorithm is a procedure or formula for solving a problem. The word derives from the name of the mathematician, Mohammed Ibn-Musa al-Khwarizmi. An algorithm is a finite step-by-step procedure to achieve a required result. A sequential solution of any program that written in human language, called algorithm. Algorithm is first step of the solution process, after the analysis of problem, programmer writes the algorithm of that problem. Algorithms have been commonly defined in simple terms as "instructions for completing a task”. Algorithms are not a special type of operation, necessarily. They are conceptual, a set of steps that you take in code to reach a specific goal.

**How To Design An Algorithm:**

Designing the right algorithm for a given application is a difficult job. It requires a major creative act, taking a problem and pulling a solution out of the ether. This is much more difficult than taking someone else's idea and modifying it to make it a little better.

When faced with a design problem, too many people freeze up in their thinking. After reading or hearing the problem, they sit down and realize that they don't know what to do next. They start panic, and finally end up settling for the first thing that comes to mind. Avoid this fate. Follow the sequence of questions provided below and in most of the catalog problem sections. The key to algorithm design (or any other problem-solving task) is to proceed by asking yourself a sequence of questions to guide your thought process.

Following are some **Steps** to design an algorithm:

* *Do I really understand the problem?*
* *What if we do this? What if we do that?*
* *What exactly does the input consist of?*
* *What exactly are the desired results or output?*
* *Can I construct an example input small enough to solve by hand? What happens when I try to solve it?*
* *How important is it to my application that I always find an exact, optimal answer? Can I settle for something that is usually pretty good?*
* *How large will a typical instance of my problem be? Will I be working on 10 items? 1,000 items? 1,000,000 items?(either it is a general solution for any number of input)*
* *How important is speed in my application? Must the problem be solved within one second? One minute? One hour? One day?*

**Example 1:**

Implement pseudo-code and algorithm that takes an input of two number s A and B and add them.

Pseudo-code

1. Input two numbers (A, B)
2. Add them ( A + B)
3. Assign in another variable “sum” (sum A+B)
4. Display that variable in which added value of both variable lie (print sum)

Algorithm Implementation

1. Start
2. Sum = 0
3. Input number A
4. Input number B
5. Sum = A+B
6. Display Sum
7. Stop

**Example 2:**

Implement pseudo-code and algorithm that takes an input of two number s A and B and multiply them.

Pseudo-code

1. Input two numbers A, B
2. Multiply them ( A \* B)
3. Assign in another variable “mult” ( mult A \* B)
4. Display that variable in which added value of both variable lie (print mult)

Algorithm Implementation

1. Start
2. multiply = 0
3. Input number A
4. Input number B
5. multiply = A\*B
6. Display multiply
7. Stop

**EXERCISES**

Q1 (a). Implement pseudo-code & algorithm that takes an input of two number s A and B and perform subtraction.

Start Start

Take two numbers Input A

Subtract them Input B

Print the result Display(A-B)

End End

Q1(b). Implement pseudo-code & algorithm that takes an input of two number s A and B and perform division.

Start Start

Take two numbers Input A

Divide them Input B

Print the result Display (A/B)

End End

Q2. Write an algorithm to find out number is odd or even?

Start Start

Take any number Input number

If number mod 2 is 0 if(number%2) ==0

Then ‘number is even’ Display (“number is even”);

Else ‘number is odd’ Else

Stop Display (“number is odd”)

Stop

Q3. Differentiate between pseudo-code and algorithm with example?

**Pseudo-code:**

Pseudocode is the statement in plain English which may be translated later into a programming language (program).

Pseudo code as i understand is an intermediary between an algorithm and implemented program. You can base your pseudo code on an algorithm. This contains transferable steps to implement.

**Algorithm:**

Algorithm on the other hand implies the overall logic. in sequential steps to solve a problem. if it is represented diagrammatically it is called a flow chart.

Algorithm you would find more common in programing terminology like systems programming, searching, sorting, basically anywhere there is a complex logic to be implemented.

Q4. Write pseudo-code and algorithm to find average of two numbers?

Start Start

Take two number Input num1, num2

Add them & divide the sum with two Display((num1+num2)/2)

Print the result Stop

Stop