Module II

Problem Solving strategies – Problem analysis – formal definition of problem – Solution – top- down design – breaking a problem into sub problems- overview of the solution to the sub problems by writing step by step procedure (algorithm) - representation of procedure by flowchart - Implementation of algorithms – use of procedures to achieve modularity. Examples for algorithms and flow charts - at least 10 problems must be discussed in detail.

Problem Solving Strategies

Can you think of a day in your life which goes without problem solving? Answer to this question is of course, No. In our life we are bound to solve problems. In our day to day activity such as purchasing something from a general store and making payments, depositing fee in school, or withdrawing money from bank account. All these activities involve some kind of problem solving. It can be said that whatever activity a human being or machine do for achieving a specified objective comes under problem solving.

There could be many problem solving strategies. The plain mater of fact is that there is no universal method. Different strategies appear to work for different people. We must start from the premise that computer problem-solving is about understanding.

As stated earlier, a program is needed to instruct the computer about the way a task is to be performed. The instructions in a program have three essential parts:

1. Instructions to accept the input data that needs to be processed,
2. Instructions that will act upon the input data and process it, and
3. Instructions to provide the output to the user

The instructions in a program are defined in a specific sequence. Writing a computer program is not a straightforward task. A person who writes the program (computer programmer) has to follow the Program Development Life Cycle.

Let’s now discuss the steps that are followed by the programmer for writing a program:

Problem Analysis : Success in solving any problem is only possible after we have made the effort to come to terms with or understand the problem at hand. In other ways we must work out what must be done than how to do it. The programmer first understands the problem to be solved. The programmer determines the various ways in which the problem can be solved, and decides upon a single solution which will be followed to solve the problem. That is the following is to be determined :

- The boundaries of the problem
- The constraints on the solution
- What actions are allowed
Formal definition of problem (Program Design) : There are many ways to solve most problems and also many solutions to most problems. In this stage we are confronted with the question “What can we do?”. The possible good solution is selected.

The selected solution is represented in a form, so that it can be coded. This requires three steps:

1. An algorithm is written, which is an English-like explanation of the solution.
2. A flowchart is drawn, which is a diagrammatic representation of the solution. The solution is represented diagrammatically, for easy understanding and clarity.
3. A pseudo code is written for the selected solution. Pseudo code uses the structured programming constructs. The pseudo code becomes an input to the next phase.

Solution

Program Development :

The pseudo code is coded using a suitable programming language. The coded pseudo code or program is compiled for any syntax errors. Syntax errors arise due to the incorrect use of programming language or due to the grammatical errors with respect to the programming language used. During compilation, the syntax errors, if any, are removed. The successfully compiled program is now ready for execution.

The executed program generates the output result, which may be correct or incorrect. The program is tested with various inputs, to see that it generates the desired results. If incorrect results are displayed, then the program has semantic error (logical error). The semantic errors are removed from the program to get the correct results. The successfully tested program is ready for use and is installed on the user’s machine.

Program Documentation and Maintenance : The program is properly documented, so that later on, anyone can use it and understand its working. Any changes made to the program, after installation, forms part of the maintenance of program. The program may require updating, fixing of errors etc. during the maintenance phase.
Top-down design: breaking a problem into sub problems

The primary goal in computer problem-solving is an algorithm which is capable of being implemented as a correct and efficient computer program. Upto algorithm design we have been mostly concerned with the very broad aspects of problem-solving.

A top-down approach (also known as stepwise design) is essentially the breaking down of a system to gain insight into the sub-systems that make it up. In a top-down approach an overview of the system is formulated, specifying but not detailing any first-level subsystems.

Definition: A solution method where the problem is broken down into smaller sub-problems, which in turn are broken down into smaller sub-problems, continuing until each sub-problem can be solved in a few steps.

Example:

Problem: An integer \( N \) is "perfect" if \( N \) is equal to the sum of the positive integers \( K \) suc that \( K < N \) and \( K \) is a divisor of \( N \). Design an algorithm to determine if a given integer is "perfect".

The top-level sub-problems (or tasks) appear to be:
1. Get the number which is to be tested.
2. Determine the divisors of the number.
3. Check if the divisors add up to the number.

Now we may consider each of these tasks separately, assuming the others will be taken care of.

Continuing our analysis:

1. Get the number which is to be tested.
   1. Output a prompt asking for the number.
   2. Input the number; call it N.
   3. Stop if N isn't positive.

2. Determine the divisors of the number.
   1. Set the divisor sum to 1. (Since 1 certainly divides N.)

Check each integer, D, from 2 \ldots N/2 to see if D is a divisor of N.

3. Check the results.
   1. If the divisor sum equals N Then,
      1. N is perfect
   2. Else
      1. N is not perfect.

Now we may consider what details need to be added to clarify the process enough to make it an algorithm.

One of the most fundamental ideas in problem solving involves subdividing a problem into subproblems and solving each subproblem individually. Quite often, the attempt to solve a subproblem introduces new subproblems at a lower level. The subdivision of subproblems should continue until a subproblem can easily be solved. This is similar to the refining of an algorithm.

In large projects, subproblems may be assigned to different programmers, or teams, who will be given precise guidelines about how that subproblem fits into the overall solution, but who will have relatively complete freedom in attacking the particular subproblem at hand.

### Advantages of Top-Down Design

- Breaking the problem into parts helps us to clarify what needs to be done.
- At each step of refinement, the new parts become less complicated and, therefore, easier to figure out.
- Parts of the solution may turn out to be reusable.
- Breaking the problem into parts allows more than one person to work on the solution.
Overview of the solution to the sub problems by writing step by step procedure

As stated earlier, problem solving is the process of solving a problem using a computer system by following a sequence of steps. The problem-solving process contains the following steps.

- Preparing hierarchy chart
- Developing algorithm
- Drawing flowchart
- Writing pseudocode

Hierarchy chart

The hierarchy chart shows the relationship of various units. It shows the top-down solution of the problem.

Algorithms

The algorithm and flowchart classification to the three types of control structures.

They are:

1. Sequence
2. Branching (Selection)
3. Loop (Repetition)

These three control structures are sufficient for all purposes. The sequence is exemplified by sequence of statements place one after the other - the one above or before another gets executed first. In flowcharts, sequence of statements is usually contained in the rectangular process box.

- The branch refers to a binary decision based on some condition. If the condition is true, one of the two branches is explored; if the condition is false, the other alternative is taken. This is usually represented by the ‘if-then’ construct in pseudo-codes and programs. In flowcharts, this is represented by the diamond-shaped decision box. This structure is also known as the selection structure.

- The loop allows a statement or a sequence of statements to be repeatedly executed based on some loop condition. It is represented by the ‘while’ and ‘for’ constructs in most programming languages, for unbounded loops and bounded loops respectively. (Unbounded loops refer to those whose number of iterations depends on the eventuality that the termination condition is satisfied; bounded loops refer to those whose number of iterations is known beforehand.) In the flowcharts, a back arrow hints the presence of a loop. A trip around the loop is known as iteration. You must ensure that the condition for the termination of the looping must be satisfied after some finite number of iterations, otherwise it ends up as an infinite loop, a common mistake made by inexperienced programmers. The loop is also known as the repetition structure.
Combining the use of these control structures, for example, a loop within a loop (nested loops), a branch within another branch (nested if), a branch within a loop, a loop within a branch, and so forth, is not uncommon. Complex algorithms may have more complicated logic structure and deep level of nesting, in which case it is best to demarcate parts of the algorithm as separate smaller modules. Beginners must train themselves to be proficient in using and combining control structures appropriately, and go through the trouble of tracing through the algorithm before they convert it into code.

**Representation of procedure by flowchart**

The flowchart is a diagram which visually presents the flow of data through processing systems. This means by seeing a flow chart one can know the operations performed and the sequence of these operations in a system. Algorithms are nothing but sequence of steps for solving problems. So a flow chart can be used for representing an algorithm. A flowchart, will describe the operations (and in what sequence) are required to solve a given problem. You can see a flow chart as a blueprint of a design you have made for solving a problem. For example suppose you are going for a picnic with your friends then you plan for the activities you will do there. If you have a plan of activities then you know clearly when you will do what activity. Similarly when you have a problem to solve using computer or in other word you need to write a computer program for a problem then it will be good to draw a flowchart prior to writing a computer program. Flowchart is drawn according to defined rules.

**Flowchart Symbols**

There are 6 basic symbols commonly used in flowcharting of assembly language Programs: Terminal, Process, input/output, Decision, Connector and Predefined Process. This is not a complete list of all the possible flowcharting symbols, it is the ones used most often in the structure of Assembly language programming.
Implementation of Algorithms

There is one rule to implement the algorithms: top to bottom rule.

Top to bottom rule: if an algorithm has been designed the path of execution should flow in a straight line from top to bottom.

Use of procedures to achieve modularity

The procedures are used to assist the implementation and readability of the main program. A set of independent procedures are to be created to perform specific and well-defined tasks. When the modularization is implemented using procedures there reaches a point where the implementation becomes difficult to read because of the fragmentation. When a large project is to be created we can choose the top-down design approach. In the first phase of the implementation, before we have implemented any of the procedures, we can just place a write statement in the skeleton procedures which simply writes out the procedure's name when it is called. This practice allows us to test the mechanism of the main program at an early stage and implement and test the procedures one by one. When a new procedure has been implemented we simply substitute it for its “dummy” procedure.
Examples for algorithms and flow charts

Problem 1: Find the area of a Circle of radius r.
Inputs to the algorithm: Radius r of the Circle.
Expected output: Area of the Circle

Algorithm:
Step 1: Read\input the Radius r of the Circle
Step 2: Area PI*r*r // calculation of area
Step 3: Print Area

Problem 2: Write an algorithm to read two numbers and find their sum.
Inputs to the algorithm: First num1. Second num2.
Expected output: Sum of the two numbers.
Algorithm:
Step 1: Start
Step 2: Read\input the first num1.
Step 3: Read\input the second num2.
Step 4: Sum num1+num2 // calculation of sum
Step 5: Print Sum
Step 6: End

Problem 3: Convert temperature Fahrenheit to Celsius
Inputs to the algorithm: Temperature in Fahrenheit
Expected output: Temperature in Celsius
Algorithm:
Step 1: Start
Step 2: Read Temperature in Fahrenheit F
Step 3: C = 5/9*(F−32)
Step 4: Print Temperature in Celsius: C
Step 5: End
**Problem 4:** write algorithm to find the greater number between two numbers

1. Start
2. Read/input A and B
3. If A greater than B then C=A
4. If B greater than A then C=B
5. Print C
6. End

**Problem 5:** write algorithm to find the result of equation

1. Start
2. Read/input x
3. If X Less than zero then F=-X
4. If X greater than or equal zero then F=X
5. Print F
6. End

**Problem 6:** A algorithm to find the largest value of any three numbers.

1. Start
2. Read/input A, B and C
3. If (A>=B) and (A>=C) then Print A go to step 7
4. If (A>=B) and (A<C) then Print C go to step 7
5. If (A<B) and (B>=C) then Print B go to step 7
6. If (A<B) and (B<C) then Print C go to step 7
7. End
Problem 7: An algorithm to calculate even numbers between 0 and 99

1. Start
2. I ← 0
3. Write I in standard output
4. I ← I+2
5. If (I <=98) then go to line 3
6. End

Problem 8: find the factorial of a number.

1. Start
2. Read n
3. Fact=n;
4. n=n-1
5. Fact=Fact*n
6. if (n>0) then go to step 4
7. Write Fact
8. End
Problem 9: to find and print the largest of N (N can be any number) numbers. Read numbers one by one.

Step 1: \textit{Input} N
Step 2: \textit{Input} X
Step 3: Max \leftarrow \text{Current}
Step 4: Counter \leftarrow 1
Step 5: \textit{While} (Counter < N)
\hspace{1cm} \text{Repeat steps 5 through 8}
Step 6: Counter \leftarrow \text{Counter} + 1
Step 7: \textit{Input} X
Step 8: \textit{If} (X > \text{Max}) \text{ then}
\hspace{1cm} \text{Max} \leftarrow X
\hspace{1cm} \textit{end if}
Step 9: \textit{Print} Max
Problem Problem 10: to print Fibonacci series

Algorithm
Step 1: Start
Step 2: Read n
Step 3: F0 = 0
Step 4: F1 = 1
Step 5: i = 1
Step 6: Fib
Step 7: Write F0, F1
Step 8: i = 1
Step 9: Fib = F0 + F1
Step 10: i = i + 1
Step 11: (If i < n)
Step 12: Stop

Yes
No