2.1.1 Problem solving and design

An algorithm is a plan, a logical step-by-step process for solving a problem. Algorithms are normally written as a flowchart or in pseudocode. The key to any problem-solving task is to guide your thought process. The most useful thing to do is keep asking ‘What if we did it this way?’ Exploring different ways of solving a problem can help to find the best way to solve it. When designing an algorithm, consider if there is more than one way of solving the problem. When designing an algorithm there are two main areas to look at:

The big picture - What is the final goal?
The individual stages – What hurdles need to be overcome on the way to the goal?

Understanding the problem

Before an algorithm can be designed, it is important to check that the problem is completely understood. There are a number of basic things to know in order to really understand the problem:
- What are the inputs into the problem?
- What will be the outputs of the problem?
- In what order do instructions need to be carried out?
- What decisions need to be made in the problem?
- Are any areas of the problem repeated?

Once these basic things are understood, it is time to design the algorithm.

Create a solution

Top-down design
A technique to any big or complex problem is to use top-down design, which means breaking down the solution to a problem or task into a number of steps that can be considered as separate sub-solution or sub-task. We can continue this process of breaking down the sub-solution into smaller solution or tasks, until we reach simple steps. A good strategy is to breakdown the solution according to the input-storage-processing-output.

Top-down design is also known as stepwise refinement.

Advantage of top-down design
- Making the design well-structured and easier to understand, modify and debug.
- Speeding up development
- Each sub-solution or module can be given to a different programmer in the team.
Example problem

Write a program to input 10 numbers and output their total

We can use a structure diagram to represent how this problem has been broken-down into sub-solutions, sequence from left to right

First two steps can be further divided into smaller sub-solutions

After breaking down the solution into smaller, more manageable sub-solutions, now we can devise an algorithm for each part
Program flowchart

The above problem can be shown the form of program flowchart

A flowchart is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

A simple flowchart representing a process for dealing with a non-functioning lamp.
Symbols used in program flowchart

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>✈️</td>
<td>Start/end</td>
<td>An oval represents a start or end point.</td>
</tr>
<tr>
<td>→</td>
<td>Arrows</td>
<td>A line is a connector that shows relationships between the representative shapes.</td>
</tr>
<tr>
<td>🔄</td>
<td>Input/Output</td>
<td>A parallelogram represents input or output.</td>
</tr>
<tr>
<td>🟨</td>
<td>Process</td>
<td>A rectangle represents a process.</td>
</tr>
<tr>
<td>✠</td>
<td>Decision</td>
<td>A diamond indicates a decision.</td>
</tr>
</tbody>
</table>

Pseudocode

Pseudocode is an informal high-level description of a computer program or algorithm. It uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are essential for machine understanding of the algorithm, such as variable declarations, system-specific code and some subroutines.

The above flowchart can be expressed in the form of pseudocode as follows

Set sum to zero
Set N to 1
Repeat
    Sum ← sum + N
    N ← N+1
Until N >= 50
Print sum
Library routines and sub-routines

In computer programming, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed. Subprograms may be defined within programs, or separately in libraries that can be used by multiple programs. In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram.

Purpose of algorithm/flowchart

Study the flowchart very carefully

What is the purpose of above flowchart?

Answer: To find the average of all positive numbers.
The following pseudocode was written to input 1000 dates.

```
1  count = 1
2  repeat
3       input day, month, year
4       count = count + 1
5  until count = 1000
```

(i) Describe why the loop only inputs 999 dates instead of 1000.

The value of count starts from 1 so only 999 iterations work.

(ii) What needs to be changed or added to the above code to make sure 1000 dates are input?

Line 1 can be changed to count=0

Or

Line 5 can be changed to (until count=1001)

Or

Until count>1000

(b) Errors in code can be found using test data.

Name three different types of test data. Using `month` from the pseudocode above, give an example of each type of test data.

- **Normal/valid (test data)**
  any value in given range (1 to 12) e.g. 4

- **Abnormal/invalid (test data)**
  any value which is outside the range/any value not acceptable i.e. letters, negative numbers, values > 12 e.g. adfrk, -20, 36

- **Extreme/boundary (test data)**
  data which is on the boundaries/edges of the acceptable range i.e. 1 or 12 for extreme; 0, 1, 12 or 13 for boundary

  Month names, instead of values, are acceptable e.g. April
Validation and verification checks on input data

Validation include (range check, length check, type check, check digit, consistency check and presence check)

Which validation methods are being carried out by the following three pseudocode statements?

(i) if age > 10 and age < 20 then print "correct"
    Range check
    Note: Range check validates upper and lower boundaries

(ii) if gender = "male" and title = "Mr" then print "correct"
    Consistency check or crossfield check

(iii) if field = "" then print "input data is missing"
    Presence check

A different application needs the whole PIN to be input.
The following code has been written to check the PIN:

```plaintext
c = 0
INPUT PIN
x = PIN
REPEAT
    x = x/10
    c = c + 1
UNTIL x < 1
IF c < 5
    THEN
        PRINT "error in PIN entered"
    ELSE
        PRINT "PIN OK"
    ENDIF
```

What type of validation check is being carried out here?

Answer: Length check
Trace tables to find the value of variables at each step in algorithm

Carefully study the following flowchart:

START

INPUT number

count = 1: total = 0: neg = 0

INPUT temp

temp >= 20 ?

Yes
total = total + temp

No

temp <= 0 ?

Yes
	neg = neg + 1

No

count = count + 1

COUNT <= number ?

Yes

STOP

No

OUTPUT total, neg

Complete the trace tables for the following two sets of test data:

(i) number = 7, temp = -5, 0, 5, -4, 0, 10, -2

(ii) number = 6, temp = 21, 20, 30, 19, 21, 15
(i) trace table:

<table>
<thead>
<tr>
<th>number</th>
<th>count</th>
<th>temp</th>
<th>total</th>
<th>neg</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(ii) trace table:

<table>
<thead>
<tr>
<th>number</th>
<th>count</th>
<th>temp</th>
<th>total</th>
<th>neg</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Identifying error in a given algorithm and suggest ways of removing these errors

A piece of pseudocode was written to input 1000 positive numbers and then output the highest and lowest numbers.

10 highest = 0
20 lowest = 0
30 for count = 1 to 100
40     input number
50     if number > highest then number = highest
60     if number < lowest then number = lowest
70     count = count + 1
80 next count
90 print highest, lowest

There are errors in the code.

Locate these errors and suggest a correction.

Answer

<table>
<thead>
<tr>
<th>description of possible error</th>
<th>suggested correction to error</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 20 lowest = 0</td>
<td>lowest = 100 (or even bigger value)</td>
</tr>
<tr>
<td>line 30 loop count is 1 to 100</td>
<td>count should be 1 to 1000 e.g. for count = 1 to 1000</td>
</tr>
<tr>
<td>line 50 number = highest</td>
<td>formula is reversed e.g. should be: highest = number</td>
</tr>
<tr>
<td>line 60 number = lowest</td>
<td>formula is reversed e.g. should be: lowest = number</td>
</tr>
<tr>
<td>line 70 count = count + 1 addition of count in a for ... to loop</td>
<td>remove line 70 from coding</td>
</tr>
</tbody>
</table>
The following algorithm should:

- input ten numbers
- output the largest number input
- output the average value of the input data

10    largest = 0
20    sum = 0
30    for x = 1 to 10
40        input x
50    if x > largest then x = largest
60    output largest
70    sum = sum + x
80    next x
90    average = sum * 10
100   output average

There are four errors in this algorithm.
Locate these errors and suggest a correction.

Answer

<table>
<thead>
<tr>
<th>error</th>
<th>correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 40: input x, using same input value as loop variable will cause problems or line 30: for x = 1 to 10</td>
<td>change loop variable e.g. for count = 1 to 10 or change input variable e.g. input number</td>
</tr>
<tr>
<td>line 50: formula is reversed</td>
<td>…. then largest = x (or largest = number)</td>
</tr>
<tr>
<td>line 60: output shouldn’t be inside the loop</td>
<td>100 output average, largest</td>
</tr>
<tr>
<td>line 90: incorrect formula</td>
<td>average = sum/10</td>
</tr>
</tbody>
</table>
2.1.2 Pseudocode

pseudocode is an informal high-level description of a computer program or algorithm.

It uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are essential for machine understanding of the algorithm, such as variable declarations, system-specific code and some subroutines.

### Conditional operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>More than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>More than or equal</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not Equal to</td>
</tr>
</tbody>
</table>

### Logical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>Both sides must be true</td>
</tr>
<tr>
<td>or</td>
<td>One side or other must be true</td>
</tr>
<tr>
<td>Xor</td>
<td>One side or other must be true but not both</td>
</tr>
<tr>
<td>Not</td>
<td>Negates truth</td>
</tr>
</tbody>
</table>
Use of conditional statements

Flow Diagram for (if…..else )statement

If [your condition here]
    Your code here
Else
    Your code here
End If
If you want to check more than one condition at the same time, you can use ElseIf.

If [your condition here]
   Your code here
ElseIf [your condition here]
   Your code here
ElseIf [your condition here]
   Your code here
Else
   Your code Here
End If
Just take a real-time example - When we want to analyze a mark lists we have to apply some conditions for grading students depends on the marks.

Following are the grading rule of the mark list:

1) If the marks is greater than 80 then the student get higher first class
2) If the marks less than 80 and greater than 60 then the student get first class
3) If the marks less than 60 and greater than 40 then the student get second class
4) The last condition is, if the marks less than 40 then the student fails.

Now here implementing these conditions in a VB program.

1. If totalMarks >= 80 Then
2.     MsgBox("Got Higher First Class ")
3. ElseIf totalMarks >= 60 Then
4.     MsgBox("Got First Class ")
5. ElseIf totalMarks >= 40 Then
6.     MsgBox("Just pass only")
7. Else
8.     MsgBox("Failed")
9. End If

Explanation of above program

Line 1 : Checking the total marks greaterthan or equal to 80
Line 2 : If total marks greater than 80 show message - "Got Higher First Class "
Line 3 : Checking the total marks greaterthan or equal to 60
Line 4 : If total marks greater than 60 show message - "Got First Class "
Line 5 : Checking the total marks greaterthan or equal to 40
Line 6 : If total marks greater than 40 show message - "Just pass only"
Line 7 : If those three conditions failed program go to the next coding block
Line 8 : If all fail shows message "Failed"
Line 9 : Ending the condition block
CASE ..........OF ......OTHERWISE........ENDCASE statement

It is quite tedious to program an extra IF statement for each extra route required. When the multiple conditions all involve a similar expression it is quicker and clearer to use a CASE....OF....OTHERWISE......ENDCASE Statement.

Consider the following program in which the user inputs a number representing a day of the week (1=Monday, 2=Tuesday...etc.) And the pseudocode algorithm assigns the name of the day to the variable DayName.

Input DayName

CASE DayNumber OF
    1: DayName ← “Monday”
    2: DayName ← “Tuesday”
    3: DayName ← “Wednesday”
    4: DayName ← “Thursday”
    5: DayName ← “Friday”
    6: DayName ← “Saturday”
    7: DayName ← “Sunday”
OTHERWISE
    WRITE “you have not entered number in the range 1 to 7”
ENDCASE

OUTPUT “Today is “, DayName
Loops

1. FOR....TO....NEXT
2. REPEAT.......UNTIL
3. WHILE.......DO.......ENDWHILE

The following illustration shows a loop structure that runs a set of statements until a condition becomes true.

1. For......to........Next
The FOR NEXT Loop, execute the loop body (the source code within For ..Next code block) to a fixed number of times.

For var=[startValue] To [endValue] [Step]
[loopBody]
Next [var]

var : The counter for the loop to repeat the steps.

startValue : The starting value assign to counter variable.

endValue : When the counter variable reach end value the Loop will stop.

loopBody : The source code between loop body

Let’s take a simple real time example, If you want to show a messagebox 5 times and each time you want to see how many times the message box shows.
1. startVal=1
2. endVal = 5
3. For var = startVal To endVal
4. show message
5. Next var

Line 1: Loop starts value from 1
Line 2: Loop will end when it reaches 5
Line 3: Assign the starting value to var and inform to stop when the var reach endVal
Line 4: Execute the loop body
Line 5: Taking next step, if the counter not reach the endVal

**While ..End While**

While .. End While Loop execute the code body (the source code within While and End while statements ) until it meets the specified condition. The expression is evaluated each time the loop is encountered. If the evaluation result is true, the loop body statements are executed.

```
While [condition]
  [loop body]
End While
```

**Condition : The condition set by the user**

1. counter = 1
2. While (counter <= 10)
3. Message
4. counter = counter + 1
5. End While

Line 1: Counter start from 1
Line 2: While loop checking the counter if it is less than or equal to 10
Line 3: Each time the Loop execute the and shows message
Line 4: Counter increment the value of 1 each time the loop execute
Line 5: End of the While End While Loop body
Repeat......until

```plaintext
repeat
  sum := sum + number;
  number := number - 2;
until number = 0;
```

**Difference between WHILE ...DO......ENDWHILE & REPEAT......UNTIL Loop Structure**

**WHILE ...DO**

1. The condition is tested before the body is executed.
2. It is possible that the body may never be executed. (This occurs if the condition is true on the first test)
3. The loop exits when the condition is false.

**REPEAT......UNTIL**

1. The condition is tested after the body is executed.
2. The body is always executed at least once.
3. The loop exits when the condition is true.

*Note: candidates are advised to try out solutions to a variety of different problems on a computer using a language of their choice, no particular programming language will be assumed in this syllabus*