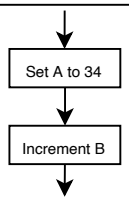
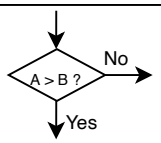
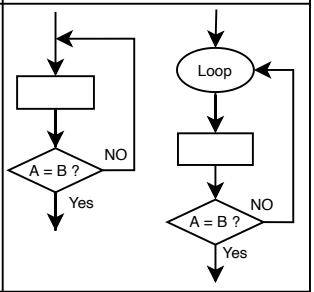
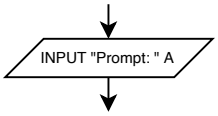
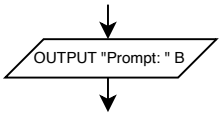
ALevel CS C12 Algorithm design and problem-solving

Structured English: a subset of the English language that consists of command statements used to describe an algorithm
Pseudocode: a way of using keywords and identifiers to describe an algorithm without following the syntax of a particular programming language
FlowChart: shapes linked together to represent the sequential steps of an algorithm

	Structure English	Pseudocode	Flowchart
Assigmen and Sequence	SET A TO 34 INCREMENT B	A ← 34 B ← B + 1	
Selection	IF A IS GREATER THAN B THEN ... ELSE...	IF A > B THEN ... ELSE ... ENDIF	
Repetition	REPEAT UNTIL A IS EQUAL TO B ...	REPEAT ... UNTIL A = B	
Input	INPUT A	INPUT "Prompt: " A	
Output	Output "Message" Output B	Output "Message", B	

Pre-condition loop:

```
SecretNumber ← Random
INPUT Guess
NumberOfGuesses ← 1
WHILE Guess <> SecretNumber DO
   IF Guess > SecretNumber
      THEN
         // the player is given the message to input a smaller number
   ENDIF
   IF Guess < SecretNumber
      THEN
         // the player is given the message to input a larger number
   ENDIF
   INPUT Guess
   NumberOfGuesses ← NumberOfGuesses + 1
ENDWHILE
OUTPUT NumberOfGuesses
```

Post-condition loop:

```
SecretNumber ← Random
NumberOfGuesses ← 0
REPEAT
   INPUT Guess
   NumberOfGuesses ← NumberOfGuesses + 1
   IF Guess > SecretNumber
      THEN
         // the player is given the message to input a smaller number
   ENDIF
   IF Guess < SecretNumber
      THEN
         // the player is given the message to input a larger number
   ENDIF
UNTIL Guess = SecretNumber
OUTPUT NumberOfGuesses
```

Logic statement:

```
IF Guess = SecretNumber
   THEN
      OUTPUT "Well done. You have guessed the secret number"
   ELSE
      IF Guess > SecretNumber AND NumberofGuesses = 10
         THEN
            OUTPUT "You still have not guessed the secret number"
         ELSE
            IF Guess > SecretNumber
               THEN
                  OUTPUT "secret number is smaller"
               ELSE
                  OUTPUT "secret number is greater"
            ENDIF
      ENDIF
ENDIF
```

REPEAT UNITL:

```
INPUT BiggestSoFar
Counter ← 1
REPEAT
   INPUT NextNumber
   Counter ← Counter + 1
   IF NextNumber > BiggestSoFar
      THEN
         BiggestSoFar ← NextNumber
   ENDIF
UNTIL Counter = 10
OUTPUT BiggestSoFar
```

For Next:

```
INPUT BiggestSoFar
FOR Counter ← 2 TO 10
   INPUT NextNumber
   IF NextNumber > BiggestSoFar
      THEN
         BiggestSoFar ← NextNumber
   ENDIF
NEXT Counter
OUTPUT BiggestSoFar
```

While loop:

```
INPUT NextNumber
BiggestSoFar ← NextNumber
WHILE NextNumber <> 0 DO // sequence terminator not encountered
   INPUT NextNumber
   IF NextNumber > BiggestSoFar
      THEN
         BiggestSoFar ← NextNumber
   ENDIF
ENDWHILE
OUTPUT BiggestSoFar
```

Procedure and Function

```
CALL SetValues

REPEAT
   CALL OutputSpaces
   CALL OutputSymbols
   NumberOfSpaces ← AdjustedNumberOfSpaces
   NumberOfSymbols ← AdjustedNumbeOfSymbols
UNTIL NumberOfSymbols > MaxNumberOfSymbols

PROCEDURE SetValues
   INPUT Symbol
   MaxNumberOfSymbols ← ValidatedMaxNumberOfSymbols
   NumberOfSpaces ← (MaxNumberOfSymbols - 1) / 2
   NumberOfSymbols ← 1
ENDPROCEDURE

FUNCTION ValidatedMaxNumberOfSymbols RETURNS INTEGER
   REPEAT
      INPUT MaxNumberOfSymbols
   UNTIL MaxNumberOfSymbols MOD 2 = 1
   RETURN MaxNumberOfSymbols
ENDFUNCTION

PROCEDURE OutputSpaces
   FOR Count1 ← 1 TO NumberOfSpaces
      OUTPUT Space // without moving to next line
   NEXT Count1
ENDPROCEDURE

PROCEDURE OutputSymbols
   FOR Count2 ← 1 TO NumberOfSymbols
      OUTPUT Symbol // without moving to next line
   NEXT Count2
   OUTPUT Newline // move to the next line
ENDPROCEDURE

FUNCTION AdjustedNumberOfSpaces RETURNS INTEGER
   NumberOfSpaces ← NumberOfSpaces – 1
   RETURN NumberOfSpaces
ENDFUNCTION

FUNCTION AdjustedNumberOfSymbols RETURNS INTEGER
   NumberOfSymbols ← NumberOfSymbols + 2
   RETURN NumberOfSymbols
ENDFUNCTION
```

ALevel CS C13 Data types and structures

Primitive data types(atomic data types): defined simply by commands built into the programming language.
Integer - whole number
Real - a number with a decimal point
TRUE or FALSE - Conditions
BOOLEAN - logical values
Char - single character

Futher data types:
String - store serveral characters
Data - A data consisting of day, month and year, sometimes including a time in hours, minutes and seconds

Recode type: know as a user-define type.

```
// declare a record type
TYPE <TypeIdentifier>
   DECLARE <field identifier> : <data type>
   .
   .
ENDTYPE
// declare a variable of this recode type
DECLARE <variable identifier> : <record type>
// access an individual filed using the dot notation
<variable identifier>.<field identifier>
```

Examples:

```
// declare a Persion record type
TYPE PersonType
   Name : STRING
   DateOfBirth : DATE
   Height : REAL
   NumberOfSiblings : INTEGER
   IsFullTimeStudent : BOOLEAN
ENDTYPE
// declare a variable of this type
DECLARE Person : PersonType
// assign a value to a file of this Person record
Person.Name ← "Fred"
Person.NumberOfSiblings ← 3
Person.IsFullTimeStudent ← TRUE
// output a field
OUTPUT Person.Name
```

Two-dimensional array:

```
// declaration
DECLARE <identifier> : ARRAY[<lBound1>:<uBound1>, <lBound2>:
<uBound2>] OF <dataType>
# example
Board : ARRAY[1:6,1:7] OF INTEGER

// Accessing 2D arrays
<arrayIdentifier>[x,y]
# example
Board[3,4] ← 0 // sets the element in row 3 and column 4 to zero

// set each element of array to zero
FOR Row ← 0 TO MaxRowIndex
   FOR Column ← 0 TO MaxColumnIndex
      ThisTable[Row, Column] ← 0
   NEXT Column
NEXT Row

// output

FOR Row ← 0 TO MaxRowIndex
   FOR Column ← 0 TO MaxColumnIndex
      OUTPUT ThisTable[Row, Column] // stay on same line
   NEXT Column
   OUTPUT Newline // move to next line for next row
NEXT Row
```

Text file

```
// writing a text file
OPENFILE <filename> FOR WRITE   //open the file for writing
WRITEFILE <filename>, <stringValue>    //write a line of text to the file
CLOSEFILE <filename> //close file

// reading from a text file
OPENFILE <filename> FOR READ // open file for reading
READFILE <filename>, <stringVariable> // read a line of text from the
file
CLOSEFILE <filename>  // close file
```

One-dimensional array:

```
// 1D array declaration
DECLARE <arrayIdentifier> : ARRAY[<lowerBound>:<upperBound>]
OF <dataType>
// pseudocode example
DECLARE List1 : ARRAY[1:3] OF STRING // 3 elements in this list
DECLARE List2 : ARRAY[0:5] OF INTEGER // 6 elements in this list
DECLARE List3 : ARRAY[1:100] OF INTEGER // 100 elements in this
list
DECLARE List4 : ARRAY[0:25] OF CHAR // 26 elements in this list
// accessing 1D array
<arrayIdentifier>[x]
// pseudocode example
NList[25] ← 0
AList[3] ← 'D'
```

iterator for loop:

```
FOR Index ← 0 TO 6
   INPUT MyList[Index]
NEXT Index
```

Linear search

```
MaxIndex ← 6
INPUT SearchValue
Found ← FALSE
Index ← –1
REPEAT
   Index ← Index + 1
   IF MyList[Index] = SearchValue
      THEN
         Found ← TRUE
   ENDIF
UNTIL FOUND = TRUE OR Index >= MaxIndex
IF Found = TRUE
   THEN
      OUTPUT "Value found at location: " Index
   ELSE
      OUTPUT "Value not found"
ENDIF
```

Bubble sort

```
n ← MaxIndex – 1
FOR i ← 0 TO MaxIndex – 1
   FOR j ← 0 TO n
      IF MyList[j] > MyList[j + 1]
         THEN
            Temp ← MyList[j]
            MyList[j] ← MyList[j + 1]
            MyList[j + 1] ← Temp
      ENDIF
   NEXT j
   n ← n – 1 // this means the next time round the inner loop, we don't
         // look at the values already in the correct positions.
NEXT i
```

improved bubble sort algorithm

```
n ← MaxIndex – 1
REPEAT
   NoMoreSwaps ← TRUE
   FOR j ← 0 TO n
      IF MyList[j] > MyList[j + 1]
      THEN
         Temp ← MyList[j]
         MyList[j] ← MyList[j + 1]
         MyList[j + 1] ← Temp
         NoMoreSwaps ← FALSE
      ENDIF
   NEXT j
   n ← n – 1
UNTIL NoMoreSwaps = TRUE
```

```
// Appending to a text file
OPENFILE <filename> FOR APPEND // open file for append
WRITEFILE <filename>, <stringValue> // write a line of text to the file
CLOSEFILE <filename> // close file

// The end-of-file(EOF) marker
OPENFILE "Test.txt" FOR READ
WHILE NOT EOF("Test.txt") DO
   READFILE "Test.txt", TextString
   OUTPUT TextString
ENDWHILE
CLOSEFILE "Test.txt"
```

ALevel CS C14 Programming (1) Java

Declare of variables:

pseudocode:
DECLARE <identifier> : <dataType>
Example
DECLARE Number1 : INTEGER // this declares Number1 to store a whole number
DECLARE YourName : STRING // this declares YourName to store a
// sequence of characters
DECLARE N1, N2, N3 : INTEGER // declares 3 integer variables
DECLARE Name1, Name2 : STRING // declares 2 string variables
Java:
int number1;
String yourName;
int n1, n2, n3;
String name1, name2;

Declare and assignment of constants:

pseudocode:
CONSTANT <identifier> = <value>
Example
CONSTANT Pi = 3.14
Java:
static final double PI = 3.14;

Assignment of variables:

pseudocode:
<identifier> ← <expression>
Example
A ← 34
B ← B + 1
Java:
A = 34;
B = B + 1;

Description	pseudocode	java
While signed numbers	INTEGER	int(4 bytes)
signed numbers with a decimal point	REAL	float (4 bytes) double(8
A single character	CHAT(single ' quotation)	char(2 bytes - Unicode)
A sequence of characters(a String)	STRING(doubl " quotation)	String (2 bytes per character)
Logical values	BOOLEAN	Boolean true false
Date value	DATE	import java.util.Date

operation	pseudocode	java
Addition	+	+
Substractior	-	-
multiplicatio	*	*
Division	/	/
Exponent	^	/
Integer division	DIV	/
Modulus	MOD	%

Operation	pseudocode	java
equal	=	==
not equal	<>	!=
greater than	>	>
less than	<	<
greater than or equal to	>=	>=
less than or equal to	<=	<=

Operation	pseudocode	java
AND (logical conjunction)	AND	&&
OR (logical inclusion)	OR	II
NOT (logical negation)	NOT	!

Outputing information:

pseudocode:
OUTPUT <string>
OUTPUT <identifier(s)>
Examples:
OUTPUT "Hello ", YourName, ". Your number is ", Number1 // newline
OUTPUT "Hello " // no new line
Java:
// syntax
System.out.print(<printlist>);
System.out.println(<printlist>);
//Examples
System.out.println("Hello " + yourName + ". Your number is " + number1);
System.out.print("Hello");

Get input from the user:

pseudocode:
INPUT "Enter a number: " A
Java:
import java.util.Scanner;
Scanner console = new Scanner(System.in);
System.out.print("Enter a number: ");
a = console.next();

Comments:

Java:
// this is a comment
// this is another comment
/* this is a multi-line comment */

Selection

pseudocode:
IF <Boolean expression>
  THEN
    <statement(s)>
ENDIF
IF <Boolean expression>
  THEN
    <statement(s)>
  ELSE
    <statement(s)>
ENDIF
# Examples
IF x < 0
  THEN
    OUTPUT "Negative"
ENDIF

IF x < 0
  THEN
    OUTPUT "Negative"
  ELSE
    OUTPUT "Positive"
ENDIF

Java:
// Syntax
if (<Boolean expression>)
  <statement>;

if (<Boolean expression>)
  <statement>;
else
  <statement>;
// Examples
if (x < 0)
  System.out.println("Negative");

if (x < 0)
  System.out.println("Negative");
else
  System.out.println("Positive");

Truncating numbers

pseudocode:
INT(x : REAL) RETURNS INTEGER
STRING _ TO _ NUM(x : STRING) RETURNS REAL
Java:
Integer.valueOf(S)
Float.valueOf(x)

CASE condition:

pseudocode:
CASE OF <expression>
  <value1> : <statement(s)>
  <value2>,<value3> : <statement(s)>
  <value4> TO <value5> : <statement(s)>
  .
  .
  OTHERWISE <statement(s)>
ENDCASE
Example:
CASE OF Grade
  "A" : OUTPUT "Top grade"
  "F", "U" : OUTPUT "Fail"
  "B".."E" : OUTPUT "Pass"
OTHERWISE OUTPUT "Invalid grade"
ENDCASE
Java:
switch (grade) {
  case 'A':
    System.out.println("Top Grade");
    break;
  case 'F': case 'U':
    System.out.println("Fail");
    break;
  case 'B': case 'C': case 'D': case 'E':
    System.out.println("Pass");
    break;
  default:
    System.out.println("Invalid grade");
}

Post-condition loops

pseudocode:
REPEAT
  <statement(s)>
UNTIL <condition>
Example:
REPEAT
  INPUT "Enter Y or N: " Answer
UNTIL Answer = "Y"
Java:
do
{
  <statement(s)>
} while <condition>;

//Examples
do
{
  System.out.print("Enter Y or N: ");
  answer = console.next();
} while (!(answer.equals("Y")));

Count-controller(FOR) loops

pseudocode:
FOR <control variable> ← s TO e STEP i // STEP is optional
  <statement(s)>
NEXT <control variable>
Example:
FOR x ← 1 TO 5
  OUTPUT x
NEXT x

FOR x = 2 TO 14 STEP 3
  OUTPUT x
NEXT x
FOR x = 5 TO 1 STEP -1
  OUTPUT x
NEXT x
Java:
for (int x = 1; x < 6; x++)
{
  System.out.print(x);
}

for (int x = 2; x < 15; x = x + 3)
{
  System.out.print(x + " ");
}
for (int x = 5; x > 0; x--)
{
  System.out.print(x + " ");
}
char[] letter = {'a', 'b', 'c'};
for (char x : letter)
{
  System.out.print(x);
}

Random number generator

Java:
import java.util.Random;
Random randomNumber = new Random();
int x = randomNumber.nextInt(6) + 1;

Pre-condition loops

pseudocode:
WHILE <condition> DO
  <statement(s)>
ENDWHILE
Example:
Answer ← ""
WHILE Answer <> "Y" DO
  INPUT "Enter Y or N: " Answer
ENDWHILE
Java:
while (<condition>)
{
  <statement(s)>;
}

//Examples
String answer = "";
while(answer.equals("Y") == false)
{
  System.out.print("Enter Y or N: ");
  answer = console.next();
}

Procedure

pseudocode:
PROCEDURE <procedureIdentifier> // this is the procedure header
  <statement(s)> // these statements are the procedure body
ENDPROCEDURE
CALL <procedureIdentifier>
Examples
PROCEDURE InputOddNumber
  REPEAT
    INPUT "Enter an odd number: " Number
  UNTIL Number MOD 2 = 1
  OUTPUT "Valid number entered"
ENDPROCEDURE
CALL InputOddNumber
Java:
void <identifier> ()
{
  <statement(s)>;
}

Functions

pseudocode:
FUNCTION <functionIdentifier> RETURNS <dataType> // function header
  <statement(s)> // function body
  RETURN <value>
ENDFUNCTION
Examples
FUNCTION InputOddNumber RETURNS INTEGER
  REPEAT
    INPUT "Enter an odd number: "
Number
  UNTIL Number MOD 2 = 1
  OUTPUT "Valid number entered"
  RETURN Number
ENDFUNCTION

ALevel CS C14 Programming (2) Java

Description	pseudocode	java
Access a single character using its position P in a string ThisString	ThisString[P] Counts from 1	ThisString.charAt(P) Counts from 0
Return the character whose ASCII value is	chr(i)	(char) i;
Returns the ASCII value of character ch	ord(ch)	(int) ch;
Returns the integer value representing the length of String S	LENGTH(S : STRING) RETURNS INTEGER	S.length();
Returns leftmost L characters from S	LEFT(S : STRING, L : INTEGER) RETURNS STRING	S.subString(0, L)
Returns rightmost L characters from S	RIGHT(S : STRING, L : INTEGER) RETURNS STRING	S.subString(S.length() - L)
Returns a string of length L starting at position P from S	MID(S : STRING, P : INTEGER, L : INTEGER) RETURNS STRING	S.subString(P, P + L)
Returns the character value representing the lower case equivalent of Ch	LCASE(Ch : CHAR) RETURNS CHAR	Character.toLowerCase(ch)
Returns the character value representing the upper case equivalent of Ch	UCASE(Ch : CHAR) RETURNS CHAR	Character.toUpperCase(ch)
Returns a string formed by converting all alphabetic characters of S to upper case	TO_UPPER(S : STRING) RETURNS STRING	S.toUpperCase()
Returns a string formed by converting all alphabetic characters of S to lower case	TO_LOWER(S : STRING) RETURNS STRING	S.toLowerCase()
Concatenate(join) two strings	S1 & S2	s = S1 + S2;

Passing parameters to procedure

pseudocode:

```
# procedure header
PROCEDURE <ProcedureIdentifier> (<parameterList>)
# parameter
BYREF <identifier1> : <dataType>
BYVALUE <identifier2> : <dataType>
```

Passing parametrs by value

```
PROCEDURE OutputSymbols(BYVALUE NumberOfSymbols : INTEGER, Symbol
: CHAR)
   DECLARE Count : INTEGER
   FOR Count ← 1 TO NumberOfSymbols
      OUTPUT Symbol // without moving to next line
   NEXT Count
   OUTPUT NewLine
ENDPROCEDURE
```

Passing parameters by reference

```
PROCEDURE AdjustValuesForNextRow(BYREF Spaces : INTEGER, Symbols :
INTEGER)
   Spaces ← Spaces - 1
   Symbols ← Symbols + 2
ENDPROCEDURE
CALL AdjustValuesForNextRow(NumberOfSpaces, NumberOfSymbols)
```

Text files

pseudocode:

```
# Writing to a text file
OPENFILE <filename> FOR WRITE // open the file for writing
WRITEFILE <filename>, <stringValue> // write a line of text to the file
CLOSEFILE <filename> // close file

# Reading from a text file
OPENFILE <filename> FOR READ // open file for reading
READFILE <filename>, <stringVariable> // read a line of text from the file
CLOSEFILE <filename>  // close file

# Appending to a text file
OPENFILE <filename> FOR APPEND // open file for append
WRITEFILE <filename>, <stringValue> // write a line of text to the file
CLOSEFILE <filename> // close file

# The end-of-file (EOF) marker
OPENFILE "Test.txt" FOR READ
WHILE NOT EOF("Test.txt") DO
   READFILE "Test.txt", TextString
   OUTPUT TextString
ENDWHILE
CLOSEFILE "Test.txt"
```

Passing parameters to subroutings

pseudocode:

```
# function header

FUNCTION <functionIdentifier> (<parameterList>) RETURNS <dataType>

FUNCTION SumRange(FirstValue : INTEGER, LastValue : INTEGER) RETURNS
INTEGER
   DECLARE Sum, ThisValue : INTEGER
   Sum ← 0
   FOR ThisValue ← FirstValue TO LastValue
      Sum ← Sum + ThisValue
   NEXT ThisValue
   RETURN Sum
ENDFUNCTION
```

Arrays

pseudocode:

```
DECLARE <arrayIdentifier> : ARRAY[<lowerBound>:<upperBound>] OF <dataType>
DECLARE List1 : ARRAY[1:3] OF STRING // 3 elements in this list
DECLARE List2 : ARRAY[0:5] OF INTEGER // 6 elements in this list
DECLARE List3 : ARRAY[1:100] OF INTEGER // 100 elements in this list
DECLARE List4 : ARRAY[0:25] OF STRING // 26 elements in this list
# accessing 1D arrays
<arrayIdentifier>[x]
# examples
NList[25] = 0 // set 25th element to zero AList[3] = "D" // set 3rd element to letter D
# creating 2D arrays
DECLARE <identifier> : ARRAY[<lBound1>:<uBound1>, <lBound2>:<uBound2>] OF
<dataType>
# examples
DECLARE Board : ARRAY[1:6, 1:7] OF INTEGER
# accessing 2D arrays
<arrayIdentifier>[x, y]
# examples
Board[3,4] ← 0 // sets the element in row 3 and column 4 to zero
```

Java:

```
String[] list1 = {"","",""};
int[] list2;
list2 = new int[5];
int[] list3;
list3 = new int[100];
String[] aList;
aList = new String[25];
int[][] board = { {0, 0, 0, 0, 0, 0, 0},
                  {0, 0, 0, 0, 0, 0, 0},
                  {0, 0, 0, 0, 0, 0, 0},
                  {0, 0, 0, 0, 0, 0, 0},
                  {0, 0, 0, 0, 0, 0, 0},
                  {0, 0, 0, 0, 0, 0, 0} }
int[][] board; board = new int[6][7];
//accessing 2D array
board[2][3] = 0;
```

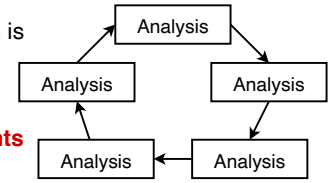
Java:

```
// Writing to a text file
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
FileWriter fileHandle = new
FileWriter("SampleFile.TXT", false);
PrintWriter printLine = new
PrintWriter(fileHandle);
String lineOfText;
printLine.printf("%s"+"%n", lineOfText);
printLine.close();

// Reading from a text file
import java.io.IOException;
import java.io.FileReader;
import java.io.BufferedReader;
FileReader fileHandle = new
FileReader("SampleFile.TXT");
BufferedReader textReader = new
BufferedReader(fileHandle);
String lineOfText = textReader.readLine();
textReader.close();

// Appending to a text file
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
FileWriter fileHandle = new
FileWriter("SampleFile.TXT"), true);
PrintWriter printLine = new
PrintWriter(fileHandle);
String lineOfText;
printLine.printf("%s"+"%n", lineOfText);
printLine.close();

// The end-of-file (EOF) marker
import java.io.IOException;
import java.io.FileReader;
import java.io.BufferedReader;
FileReader fileHandle = new FileReader("Test.
txt");
BufferedReader textReader = new
BufferedReader(fileHandle);
String lineOfText = textReader.readLine();
while (lineOfText != null)
{
   System.out.println(lineOfText);
   lineOfText = textReader.readLine();
}
textReader.close();
```

ALevel CS C15 Software Development (1)

Develop life cycle

Analysis:



1. The first step in solving a problem is to **investigate the issues** and the current system if there is one.
The problem needs to be defined clearly and precisely. A '**requirements specification**' is drawn up.
2. The next step is **planning a solution**. Sometimes there is more than one solution. You need to **decide which is the most appropriate**.
3. The third step is to **decide how to solve the problem**
   1. **bottom-up**: start with a small sub-problem and then build on this
   2. **top-down**: stepwise refinement using pseudocode, flowcharts or structure charts.

Design:
Plan your algorithm by drawing a flowchart or writing pseudocode.

Coding:
You implement your algorithm by converting your pseudocode into program code. When you start writing programs you might find it takes several attempts before the program compiles.

Testing:
Only thorough testing can ensure the program really works under all circumstances

Rapid Application Development (RAD) model

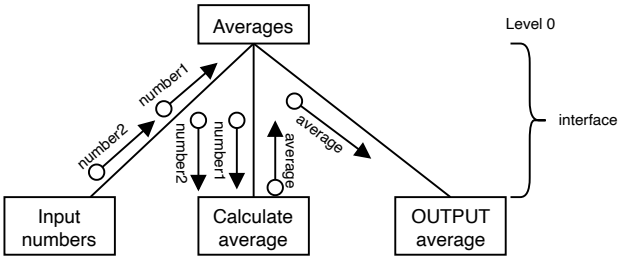
RAD is a software development methodology that uses minimal planning. Instead it uses prototyping. A prototype is a working model of part of the solution.
In the RAD model, the modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. There is no detailed preplanning. Changes are made during the development process.
The analysis, design, code and test phases are incorporated into a series of short, iterative development cycles.

Benefits:
1. Changing requirements can be accommodated.
2. Progress can be measured.
3. Productivity increases with fewer people in a short time.
4. Reduces development time.
5. Increases reusability of components.
6. Quick initial reviews occur.
7. Encourages customer feedback.
8. Integration from very beginning solves a lot of integration issues.

Drawbacks:
1. Only systems that can be modularised can be built using RAD.
2. Requires highly skilled developers/designers.
3. Suitable for systems that are component based and scalable.
4. Requires user involvement throughout the life cycle.
5. Suitable for projects requiring shorter development times.

Structure chart

top-level box is the name of the module
The arrow show how the parameters are passed between the modules. This parameter passing is known as the 'interface'.



Waterfall model

The arrows going down represent the fact that the results from one stage are input into the next stage. The arrows leading back up to an earlier stage reflect the fact that often more work is required at an earlier stage to complete the current stage.

Benefits:
1. Simple to understand as the stages are clearly defined.
2. Easy to manage due to the fixed stages in the model. Each stage has specific outcomes.
3. Stages are processed and completed one at a time.
4. Works well for smaller projects where requirements are very well understood.

Drawbacks:
1. No working software is produced until late during the life cycle.
2. Not a good model for complex and object-oriented projects.
3. Poor model for long and ongoing projects.
4. Cannot accommodate changing requirements.
5. It is difficult to measure progress within stages.
6. Integration is done at the very end, which doesn't allow identifying potential technical or business issues early.

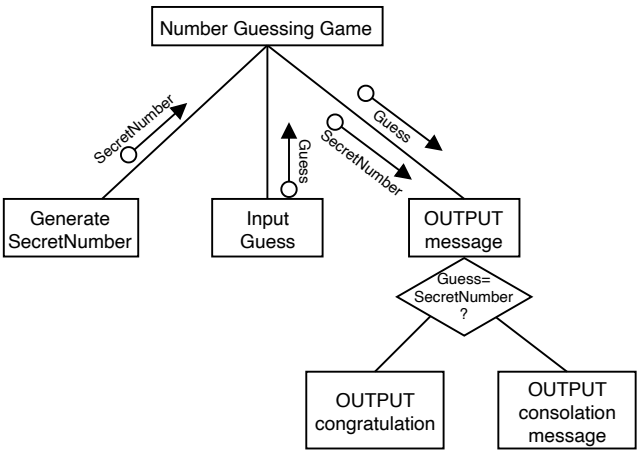
Iterative model

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development starts with the implementation of a small subset of the program requirements. Repeated (iterative) reviews to identify further requirements eventually result in the complete system.

Benefits:
1. There is a working model of the system at a very early stage of development, which makes it easier to find functional or design flaws. Finding issues at an early stage of development means corrective measures can be taken more quickly.
2. Some working functionality can be developed quickly and early in the life cycle.
3. Results are obtained early and periodically.
4. Parallel development can be planned.
5. Progress can be measured.
6. Less costly to change the scope/requirements.
7. Testing and debugging of a smaller subset of program is easy.
8. Risks are identified and resolved during iteration.
9. Easier to manage risk – high-risk part is done first.
10. With every increment, operational product is delivered.
11. Issues, challenges and risks identified from each increment can be utilised/applied to the next increment.
12. Better suited for large and mission-critical projects.
13. During the life cycle, software is produced early, which facilitates customer evaluation and feedback.

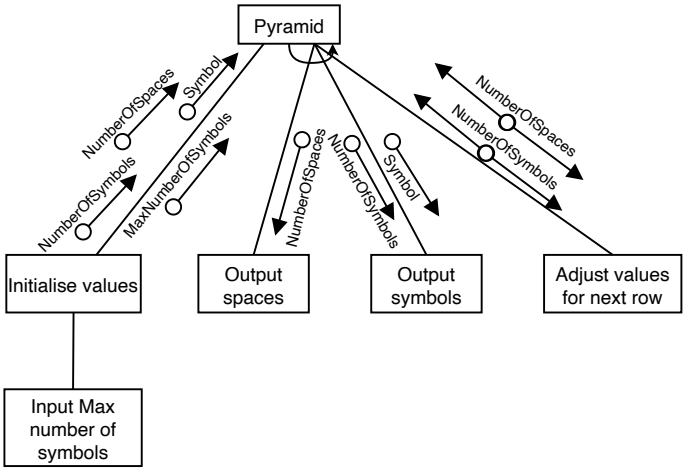
Drawbacks:
1. Only large software development projects can benefit because it is hard to break a small software system into further small serviceable modules.
2. More resources may be required.
3. Design issues might arise because not all requirements are gathered at the beginning of the entire life cycle.
4. Defining increments may require definition of the complete system.

selection



ALevel CS C15 Software Development (2)

repetition
An arrow with a solid round end ○→ show s that the value transferred is a flag (a boolean value)
A double-headed arrow ←○→ shows that the variable value is updated within the module.



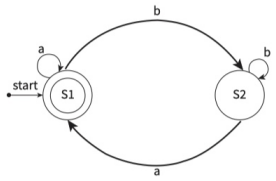
Pyramid pseudocode

```
MODULE Pyramid
   CALL SetValues(NumberOfSymbols, NumberOfSpaces, Symbol,
MaxNumberOfSymbols)
   REPEAT
      CALL OutputSpaces(NumberOfSpaces)
      CALL OutputSymbols(NumberOfSymbols, Symbol)
      CALL AdjustValuesForNextRow(NumberOfSpaces, NumberOfSymbols)
   UNTIL NumberOfSymbols > MaxNumberOfSymbols
ENDMODULE

PROCEDURE SetValues(NumberOfSymbols, NumberOfSpaces, Symbol,
MaxNumberOfSymbols)
   INPUT Symbol
   CALL InputMaxNumberOfSymbols
   NumberOfSpaces ← (MaxNumberOfSymbols – 1) / 2
   NumberOfSymbols ← 1
ENDPROCEDURE

PROCEDURE InputMaxNumberOfSymbols(MaxNumberOfSymbols)
   REPEAT
      INPUT MaxNumberOfSymbols
   UNTIL MaxNumberOfSymbols MOD 2 = 1
ENDPROCEDURE

PROCEDURE OutputSpaces(NumberOfSpaces)
   FOR Count ← 1 TO NumberOfSpaces
      OUTPUT Space // without moving to next line
   NEXT Count
ENDPROCEDURE

PROCEDURE OutputSymbols(NumberOfSymbols, Symbol)
   FOR Count← 1 TO NumberOfSymbols
      OUTPUT Symbol // without moving to next line
   NEXT Count
   OUTPUT Newline // move to the next line
ENDPROCEDURE

PROCEDURE AdjustValuesForNextRow(NumberOfSpaces, NumberOfSymbols)
   NumberOfSpaces ← NumberOfSpaces – 1
   NumberOfSymbols ← NumberOfSymbols + 2
ENDPROCEDURE
```

Test strategy:
1. flow of control: does the user get appropriate choices and does the chosen option go to the correct module?
2. validation of input: has all data been entered into the system correctly?
3. do loops and decisions perform correctly?
4. is data saved into the correct files?
5. does the system produce the correct results?

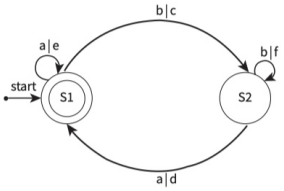
Finite state machine (FSM): a machine that consists of a fixed set of possible states with a set of inputs that change the state and a set of possible outputs
State-transition table: a table that gives information about the states of an FSM
State-transition diagram: a diagram that describes the behaviour of an FSM

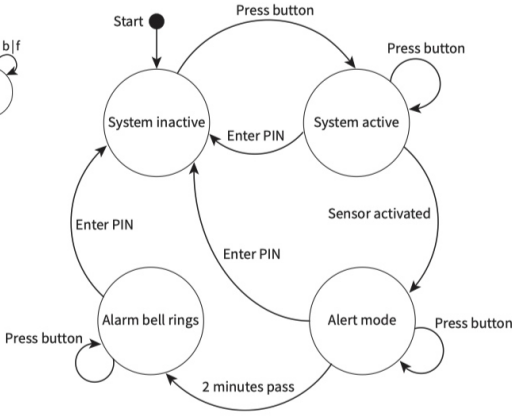
state-transition diagram



		current state	
		S1	S1
input	a	S1	S1
	b	S2	S2

state-transition diagram with outputs





Current state	Event	Next state
System inactive	Press start button	System active
System active	Enter PIN	System inactive
System active	Activate sensor	Alter mode
System active	Press start button	System active
Alert mode	Enter PIN	System inactive
Alert mode	2 minutes pass	Alarm bell ringing
Alert mode	Press start button	Alert mode
Alarm bell ringing	Enter PIN	System inactive
Alarm bell ringing	press start button	Alarm bell ringing

Syntax error: an error in which a program statement does not follow the rules of the language
Logic error: an error in the logic of the solution that causes it not to behave as intended
Run-time error: an error that causes program execution to crash or freeze
Why errors occur and how to find them:
1. the programmer has made a coding mistake
2. the requirement specification was not drawn up correctly
3. the soft ware designer has made a design error
4. the user interface is poorly designed, and the user makes mistakes
5. computer hardware experiences failure.
Test data: carefully chosen values that will test a program
Black-box testing: comparing expected results with actual results when a program is run
White-box testing: testing every path through the program code
Dry-run (walk through): the process of checking the execution of an algorithm or program by recording variable values in a trace table
Trace table: a table with a column for each variable that records their changing values
Integration testing: individually tested modules are joined into one program and tested to ensure the modules interact correctly
Alpha testing: testing of software in-house by dedicated testers
Acceptance testing: testing of software by customers before sign-off
Beta testing: testing of software by a limited number of chosen users before general release