Lecture # 8

Structured English/ Pseudocode

Introduction to Programming

Program development

- Development of a computer program can be broken down broadly into 5 stages
    1. Problem analysis
    2. Designing and testing the algorithm
    3. Coding the algorithm
    4. Testing the code
    5. Documentation

1. Problem analysis

- Involves studying the problem, in order to understand the nature of the problem and determine how to solve it

2. Designing and testing the algorithm

- An ***algorithm*** is a solution to a problem consisting of a series of steps
- The ***algorithm*** may be represented by a *flowchart* or a narrative of the solution known as *pseudocode*
- Having designed a solution, the next step is to trace through the ***algorithm*** with test data to verify that the solution contains no ***logical errors***.
- ***Logical errors*** are mistakes in the design of a program

3. Coding the algorithm

- Use a suitable computer language to code the algorithm
- The operations defined in the pseudocode or flowchart should translate directly into instructions in a high-level language

4. Testing the code

- The program must be compiled
  - The compiler will list any ***syntax errors***
  - syntax errors are errors in the way the grammar of the language has been used
- The program will then be tested to determine whether it meets the original specification (usually tracing or peer-group inspection)
- The program is then tested using suitable test data at run time

5. Documentation

- Documentation is used and produced during all stages of program lifecycle
- Documentation involves
  - Documenting the purpose of the program
  - The method of the solution
  - The stages of testing that it has undergone

Algorithms

- Algorithm – a *procedure* for solving a problem in terms of
  - the *actions* to be executed and
  - the *order* in which these actions are to be executed

Algorithm

- An algorithm should be
  - – Complete (covers everything)
  - – Unambiguous (no doubt)
  - – Deterministic (only one possible result)
    - If same data is input same result given
  - – Finite (it should finish)

Examples of algorithms

- Directions to get from A to B
- A musical score
- A recipe
- A set of instructions to solve the Rubix cube
- A set of instructions to tell a computer to perform a given task – computer program

Write an algorithm to solve problem

- You have two containers, one holds 5 litres, and the other 3 litres. You have access to a water tap. There are no markings on either container. Devise an algorithm to place 4 litres of water in the 5 litre container, without using any other equipment.

Solution

- Fill 5 litre container
- Fill 3 litre container from the 5 litre container (leaves 2 litres in 5 litre)
- Empty 3 litre container
- Pour the 2 remaining litres from 5litre container into the 3 litre container
- Fill the 5 litre container
- Fill the 3 litre container from the 5 litre container
  - (takes 1 litre from the 5 litre, leaving 4 litres in it)

Pseudocode/ Structured English

- Pseudocode/Structured English
  - An artificial and informal program development aid that helps programmers to develop algorithms.
  - Converted to java programs
  - Consists only of action statements
  - No declarations

Structured English defines an **Algorithm**

- Statements can not be Ambiguous
- Statements can not be language dependant
- Each instruction must be “simple”
- Algorithm must have input
- Algorithm must have output

Structured English is not Language dependant

- Avoid using language statements
- Use indentation rather than ‘brackets’ to indicate structure
- There is no single definition of what Structured English/pseudocode is, it is not a ‘language’
- Style needs to be applied consistently throughout definition

Constructing Structured English

- The three primary programming constructs are used:
  - **Sequence**:Actions take place one after the other
  - **Selection**:Actions take place if a condition true
  - **Iteration**:Actions may take place several times

Sequence

- Each sequence statement must perform one operation
- The statement must be an action-oriented verb applied to an object

- Verb examples include:
  - GET ACCEPT READ PUT DISPLAY WRITE FIND SEARCH LOCATE ADD SUBTRACT MULTIPLY DIVIDE COMPUTE DELETE FIND VALIDATE MOVE REPLACE SET SORT

Sequence (examples)

GET hours worked

GET pay rate

CALC pay

DISPLAY pay

```
import java.util.Scanner;
public class CalcWage{
   public static void main(String[] args){
        //create instance of Scanner class
        Scanner keyboardIn = new Scanner(System.in);

        //declare variables
        int hoursWked;
        double rate;
        double wages;

        //Get hours worked
        System.out.print("Enter Hours worked: ");
        hoursWked = keyboardIn.nextInt();
        //GET pay rate
        System.out.print("Enter rate of pay per hour: ");
        rate = keyboardIn.nextDouble();

        //calc pay
        wages = hoursWked * rate;

        // DISPLAY gross pay
        System.out.println("Wages due:  " + wages);
   } //end main method
} //end class
```

Selection

- Selection implies choice
- Statements executed IF condition is true
- Relational/equality operators
- Logical operators (AND, OR, NOT) may be used
- **IF** condition **THEN** statement(s)
- **IF** hours worked > 40

    THEN COMPUTE overtime hours

Selection

```
IF condition
THEN statement(s)
ELSE statement(s)

IF Course mark is greater than or equal to 40
THEN
    Course Grade is “Pass”
ELSE
    Course Grade is “Fail”
```

Iteration

- Two types of iterations:
  - Number of iterations is known
  - Number of iterations is unknown

Iteration

- **FOR** each item **DO**
  statement(s)
- **FOR** each Month **DO** (no. of months is known)
  READ Sales
  COMPARE Sales to Quota
  COMPUTE Bonus

Iteration

- **WHILE** condition **DO**
  statement(s)
- **WHILE** Water Temperature is too “Hot” **DO**
  ADJUST Cold Water
  ADJUST Hot Water

Iteration

- **REPEAT**

  statement(s)

  UNTIL condition
- **REPEAT**

  PROCESS Student Record

  UNTIL No More Students

***Java does not have REPEAT/UNTIL structure

Problem 1

Write a program that will calculate the total expenses payable to an employee. The program should ask the user to enter the miles travelled and the allowance per mile. If the number of miles travelled is more than 150 then an overnight allowance of €100 should be included in the total expenses awarded. If the total expenses awarded is less than €50 add on an additional 6%.

Suggested solution

```
GET miles
GET allowPerMile
CALC expenses

IF miles > 150
    ADD 100 to expenses
IF expenses < 50
    CALC extraExpenses  (6% of expenses)
    ADD extraExpenses to expenses

DISPLAY expenses
```

Problem 2

- Write a program that will ask the user to enter hours worked and rate of pay for an employee and will calculate and display the employee’s wages. If the employee has worked overtime (over 38 hours) allow them time and a half for the extra hours worked.

Suggested Solution

```
GET hours
GET rate

CALC wages (hours * rate)

IF hours > 38
    CALC overtime  ((hours - 38) * 0.5 * rate)
    ADD overtime to wages

DISPLAY wages
```

Another Suggested Solution

```
GET hours
GET rate

IF hours <= 38
   CALC wages (hours * rate)
ELSE
    CALC overtime  ((hours - 38) * 1.5 * rate)
    CALC wages (38 * rate + overtime)

DISPLAY wages
```

Pseudocode

- Should be readable and simple
- Cannot be compiled
- Describes steps needed to solve problem
- Can be copied into source code as comments

- Used to develop and document algorithms
- Written in natural language
- Statements may begin with java keywords ie FOR each, or IF
- Indentation is used to show grouping of operations (not brackets)

- Divide complex operations into simple, more manageable tasks
- Refine until they are simple to understand and implement
- Include pseudocode as comments in your source code

Practical Question

- Write a program that will ask the user to enter two integer numbers. The user should then be asked to enter the product of the 2 numbers entered (the result of one multiplied by the other). If the user enters the correct answer an appropriate message should be displayed. If the answer entered is incorrect the user should be continually asked to enter another answer until the correct answer is entered.

Pseudocode

```
GET no1
GET no2
CALC product (no1*no2)

DO
    GET guess
    IF guess is not equal to product
        DISPLAY error message
WHILE guess is not equal to product

DISPLAY product
```