Software Development
Intermediate 2

Revision

Software Development Cycle

- Analysis
- Design
- Implementation
- Testing
- Documentation
- Evaluation
- Maintenance

- A
- Dance
- In
- The
- Dark
- Every
- Monday

Analysis

- Work out exactly what your program has to do
- Identify the inputs and outputs and process required
- Produce the **specification**

Design

- Plan the User Interface – the messages, prompts, buttons that will be required on screen

- Plan the ALGORITHM – the steps that must be performed to do the processing

Pseudocode

1. Get Input
2. Perform Calculations
3. Display output

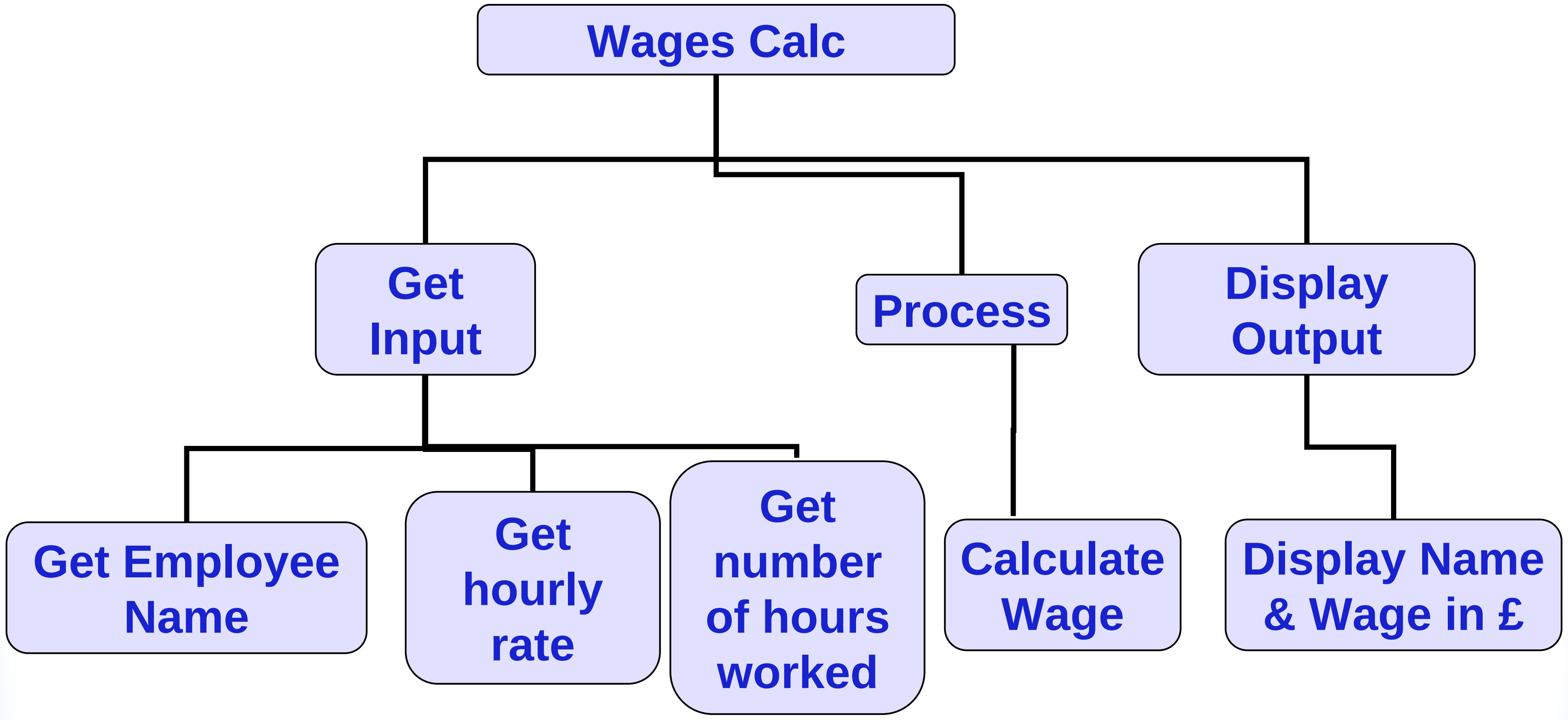
Pseudocode

```
1.  Get Input
    1.1  Get employee name
    1.2  Get hourly rate
    1.3  Get number of hours worked
2.  Perform Calculations
    2.1  Calculate wage
3.  Display output
    3.1  Display employee name
    3.2  Display wage in £
```

Pseudocode - Refinement

```
1. Get Input
   1.1 Get employee name
   1.2 Get hourly rate
   1.3 Get number of hours worked
2. Perform Calculations
   2.1 Calculate wage
      2.1.1 Multiply hourly rate by hours worked
3. Display output
   3.1 Display employee name
   3.2 Display wage in £
```

Structure Diagram



Implementation

- Writing the actual program code
- It will be based on the design

- Good pseudocode will help when writing the program as all the steps have already been planned

Testing

- Checking that the program works under most conditions

- Test Data
  - Normal
  - Extreme
  - Exceptional

Test Data

- **Normal**
  - Data that the program should expect to handle
- **Extreme**
  - Data on the limits of what would be expected
- **Exceptional**
  - Input which it has NOT been designed to cope with

Test Data

- Dice Throw – test data
- Normal
  - 2, 5
- Extreme
  - 1, 6
- Exceptional
  - 0, -3, 8, Fred, &

Documentation

- User Guide
  - How to use the program
  - May include a tutorial
  - May be a book or CD
- Technical Guide
  - Specification of computer required
  - Installation instructions

Evaluation

- Is the software “good enough” to be distributed or sold?

- Fit for purpose
- User Interface
- Readability

Evaluation

- Fit for purpose
  - Does it do all the things it was supposed to do
  - Does it meet all the requirements in the program specification
  - Does it produce the correct results

Evaluation

- User Interface
  - Is it easy to use?
  - Is it clear what the buttons, menus etc do?
  - Are any instructions to the user clear and easy to understand?
  - Is text written in a large and clear font?
  - Are colours used appropriately?

Evaluation

- Readability
  - Of NO interest to the user
  - Important for any PROGRAMMER who may need to work on the program

Evaluation

- Readability
  - Can the code be easily read and understood by another programmer
  - Objects and Variables have meaningful names
  - Code contains COMMENTS explaining what it is supposed to do
  - Code is set out in a readable way
    - Indentation is used
    - Blank lines are left between sections

Maintenance

- Occurs AFTER the program has been distributed

- Corrective
- Perfective
- Adaptive

Maintenance

- Corrective
  - Fix any errors which are discovered
- Perfective
  - Add new features
- Adaptive
  - Change to take account of new conditions
  - To run under a new operating system
    - New version for WinXP or MacOSX
  - To run on new or different hardware
    - Transfer a Windows program to Mac computers

Software Development Cycle

- Analysis
- Design
- Implementation
- Testing
- Documentation
- Evaluation
- Maintenance

- A
- Dance
- In
- The
- Dark
- Every
- Monday

Programming Languages

Levels of Language

- Machine Code
  - 10011100 10100011
- Assembler
  - MOV si, 0
- High Level Language
  - Ruby, Visual Basic, LiveCode, FORTRAN
- Macro
  - Record series of keystrokes, replay with single key combination

Machine Code

- Written in 1’s and 0’s
- Very hard to understand for humans
- Very hard to spot errors, make changes
- Specific to a processor
- Can be run by the processor

High Level Language

- Written in English-like language
- Easy to learn
- Easy to understand, make changes, spot errors
- Typed using a Text Editor
- Can be transferred between different types of computer
- Must be TRANSLATED into machine code before it is run

Translators

- **Interpreter**
- Changes the program a line at a time and then executes that line, then moves on to the next line
- Slow – as must wait for next line to be translated before moving on
- Some lines may end up being translated lots of times e.g. in a loop

Translators

- **Compiler**
- Processes the whole program and creates a file of the translated machine code
- Fast – once the machine code has been created it can be saved and re-run without being translated again
- If there are errors in the program or part of it is not yet written then the program will not compile so you cannot test part of it until it is complete and correct

Macro

- A series of steps which are recorded and can then be re-run using a single keystroke
- Used in many application programs to perform complex repetitive tasks

Programming Language Constructs

Input

- Ask

Output

- Put

Variables

- Integer

  1, 78, 0, -45, 67823

- Single (or Real)

  1.5, 94.2, 0.0, 0.000134, -4.874, 85617.23

- String

  Fred, Monday, 1st, Car99, a+%$!

Assigment and Arithmetic

- Put 1 into Answer
- Put "Monday" into Day

- Put Count + 1into Total
- Put Price * Number into Cost
- Put Total / 100 into Grade
- Put Number ^ 2 into Square

Arithmetic

+	Add
-	Subtract
*	Multiply
/	Divide
^	to the power of e.g. 4 ^ 2 = 16

- Normal Arithmetic rules apply
- BODMAS
  (Brackets, Of, Divide, Multiply, Add, Subtract)

Logical Operators

- and
- or
- not

Loops

- Fixed Loops
- Where we know at the start how many times we must repeat

- Repeat with loop = 1 to 5

```
Repeat with loop = 1 to 5
    put "Hello" after field "output"
 End Repeat
```

Loops

- We may NOT know how many times when we write the program – but we WILL know by the time we are running it

- ask "How Many pupils?"
- Put it into pupils

- Repeat with loop = 1 to pupils
-     put " Hello" after field "output"
- End repeat

Loops

- Ask "How Many pupils?"
- Put it into pupils

- Repeat with loop = 1 to pupils
-     put loop & " Hello" & return after field "output"
- End repeat

Nested Loops

- One loop INSIDE another

```
• Repeat with table = 1 to 10
•     repeat with number = 1 to 12
•        put table & " * " & number after field "output"
•     End repeat
• End repeat
```

Conditional Loops

- We don’t know how many times we must repeat – we need to keep going UNTIL something changes

- dice = 0
- Repeat until dice >= 1 and dice <= 6
-   ask "Enter a dice throw"
-   put it into dice
- End repeat

Conditions

- IF ...

- if menu = 1
  put “Hello” into field "output"
  end if

- if menu = 2
  put “Goodbye” into field "output"
  end if

Conditions

- IF ...

- if age < 5 then

```
if    age  <  5 then
    put  "You can travel free" into field "output"
end if
```

Complex Conditions

- IF ...

- if age < 5 **or** age > 65 then

```
if    age  <  5    or    age > 65  then
      put “You can travel free” into field "output"
end if
```

Complex Conditions

- IF ...

- if **not** (age > 5 **or** age < 65)
    put "You can travel free“
  end if

Complex Conditions

IF … ELSE ... END

```
• if age < 5    then
        put "You can travel free"
  else
        if age > 65   then
              put "You can get a pass"
        else
              put "You must pay the fare"
        end if
End if
```

Variables

- Integer

  1, 78, 0, -45, 67823

- Float (or Real)

  1.5, 94.2, 0.0, 0.000134, -4.874, 85617.23

- String

  Fred, Monday, 1st, Car99, a+%$!

Array

- A collection of similar variables

- Integer Array
- String Array

- Has an INDEX to identify the elements

String Array

1	Fred
2	Andrew
3	Graham
4	Billy
5	John

Names[2] = “Andrew”

Names[5] = “John”

Pre Defined Numerical Functions

- Abs
- Round
- NumToChar

Pre Defined String Functions

- ToUpper
- Length
- Offset

Standard Algorithms

Find Maximum

- Loop
  - Read in value
  - Compare to largest so far
  - If bigger, store as largest
- End Loop
- Display Largest

Find Maximum

```
Repeat with loop =  1  to  6
        ask “Enter a number”
        put it into value
        if value > largest
                largest = value
        end if
End repeat
put "The largest was " & largest into field "output"
```

Find Minimum

- Initialise smallest to a very big value (or the first actual value)
- Loop
  - Read in value
  - Compare to smallest so far
  - If smaller, store as smallest
- End Loop
- Display smallest

Find Minimum

```
Smallest = 9999
Repeat with loop = 1 to 6
        ask “Enter a number”
        put it into value
        If value < smallest
                smallest = value
        end if
End repeat
puts "The smallest was " & smallest into field "output"
```

Input Validation

- Loop
  - Prompt user for input
  - Read in input
- Until valid input given
- Process

Input Validation

- Loop
  - Prompt user for input
  - Read in input
  - If input is INVALID then tell user
- Until valid input given
- Process

Dice Throw

```
put 0 into dice
repeat until dice >= 1 and dice <= 6
        ask "Enter a dice throw: 1 to 6"
        put it into dice
        if dice < 1 or dice > 6 then
            answer "Sorry but a dice throw must be 1-6"
        end if
  end repeat
```

Face Card

```
put "" into face
repeat until face = "K" or face = "Q" or face = "J"
    ask "Enter a face card (K, Q or J)"
    put it into face
    if face <> "K" and face <> "Q" and face <> "J" then
        answer "Sorry but face card must be K, Q or J"
    end if
end repeat
```

Count Occurrences

- Loop
  - Read in value
  - If value matches then add 1 to Count
- Until all data input
- Display Count

Linear Seach

- Read in values and store in an array
- Loop
  - Check next element in array
  - If value matches then display
- Until value found or end of array reached