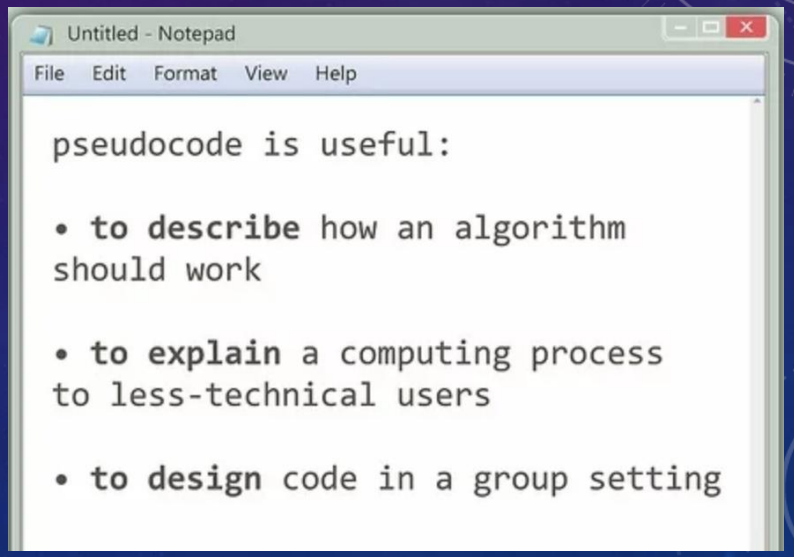
ALGORITHMS PLANNING & REFINEMENT

VISUAL BASIC

TABLE OF CONTENTS

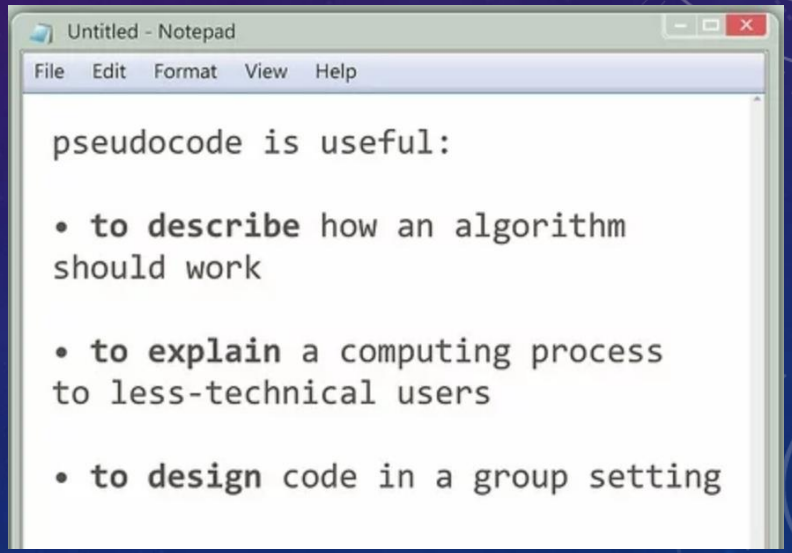
LECTURE CONTEXT

- Be familiar with pseudo code and flowcharts
- Understand the fundamentals of algorithmic design.
- Appreciate the importance of careful planning and modularity
- Know how to use top-down and stepwise refinement design techniques



LECTURE CONTEXT

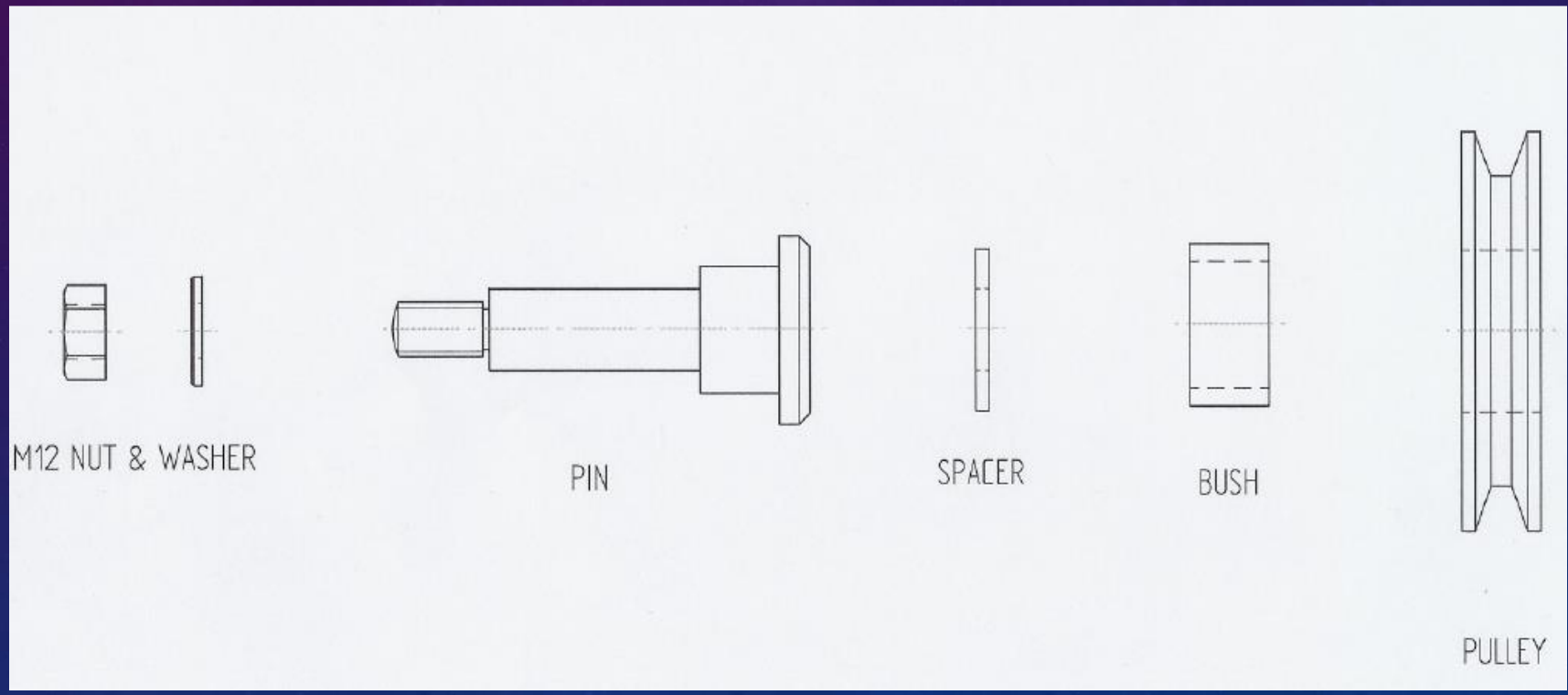
- As with all engineering tasks, good design requires good understanding of the problem and solution, as well as good planning and management of the project.
- It is also important to communicate ideas well whether in the form of an engineering drawing or an algorithm/flowchart.
- This lecture looks at some rudimentary software design and planning techniques that will make writing actual code much easier and will be required and assessed during class.



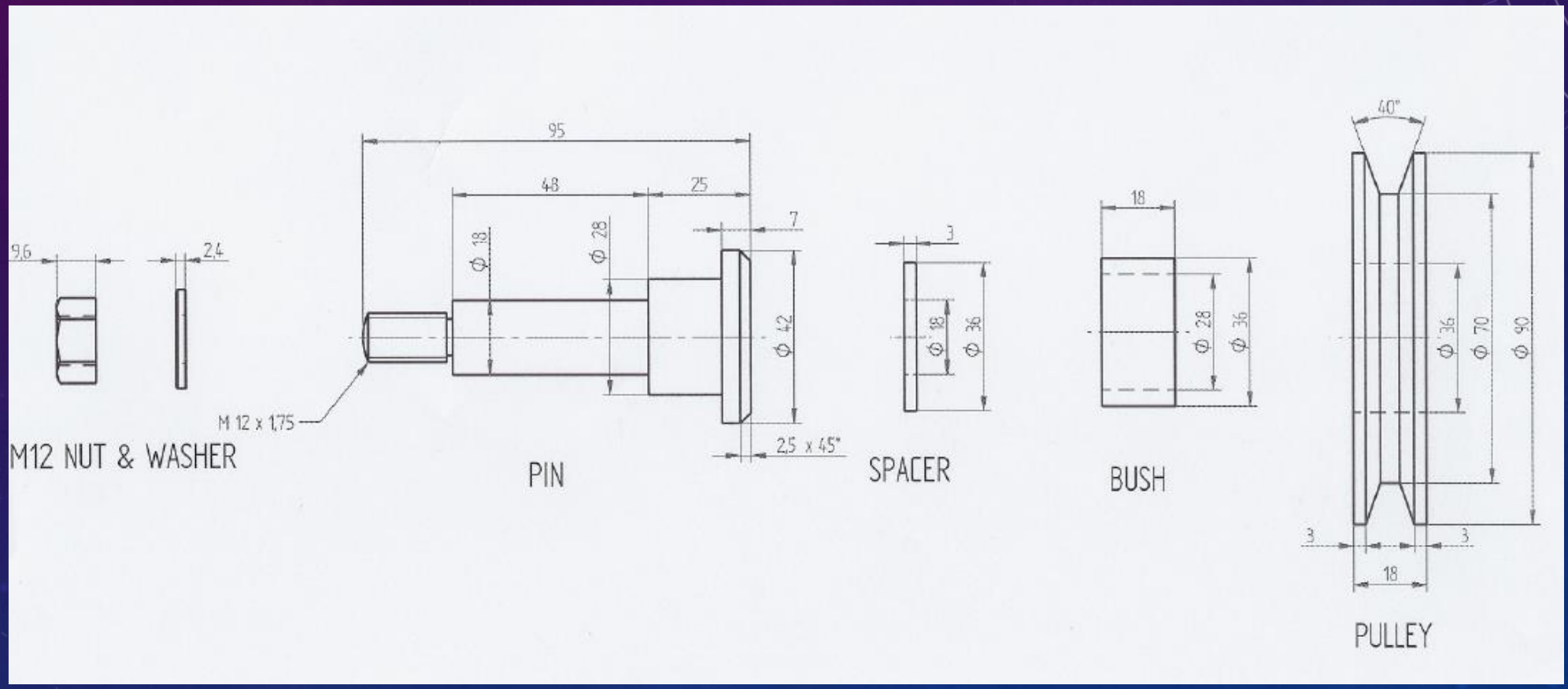
WHY SHOULD I PLAN

- Would you ask a technician to make a component without engineering drawings?
- Or without tolerances?
- So why try to write a program without planning and documentation (algorithm)?
- Students often attempt to code without really understanding the problem or flowcharting.
- Result? many errors, wasted time, stress, anger, frustration, and low marks of course.

CAN YOU MAKE THIS

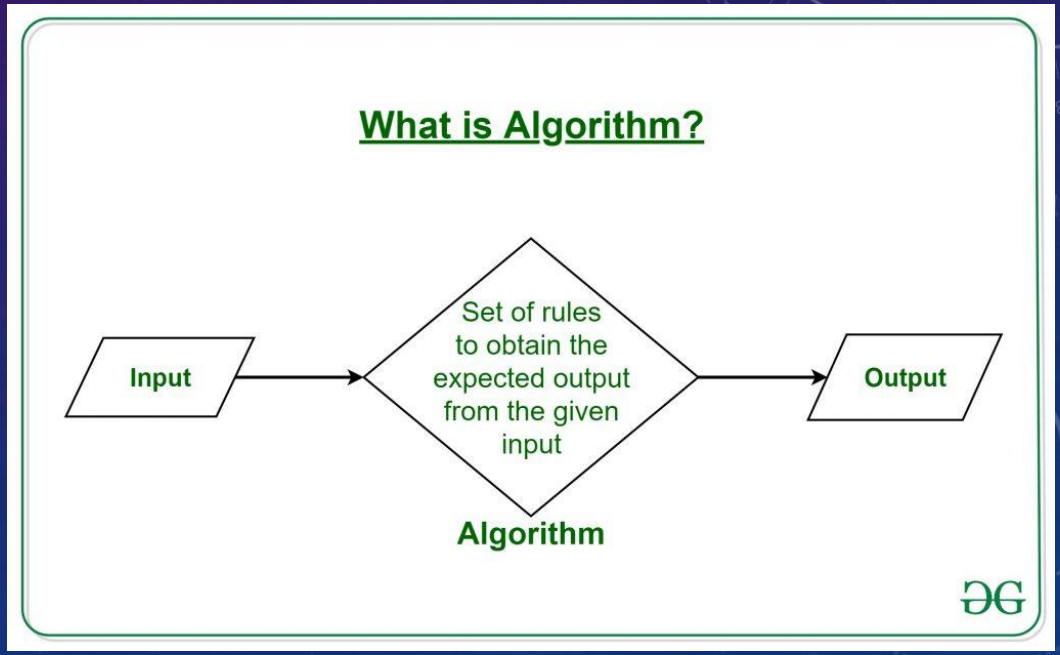


OR THIS


9,6
2,4
M12 NUT & WASHER
95
48
25
7
Φ 18
Φ 28
Φ 42
M 12 x 1,75
2,5 x 45°
PIN
3
Φ 18
Φ 36
SPACER
18
Φ 28
Φ 36
BUSH
40°
Φ 36
Φ 70
Φ 90
3
3
18
PULLEY

WHAT IS AN ALGORITHM?

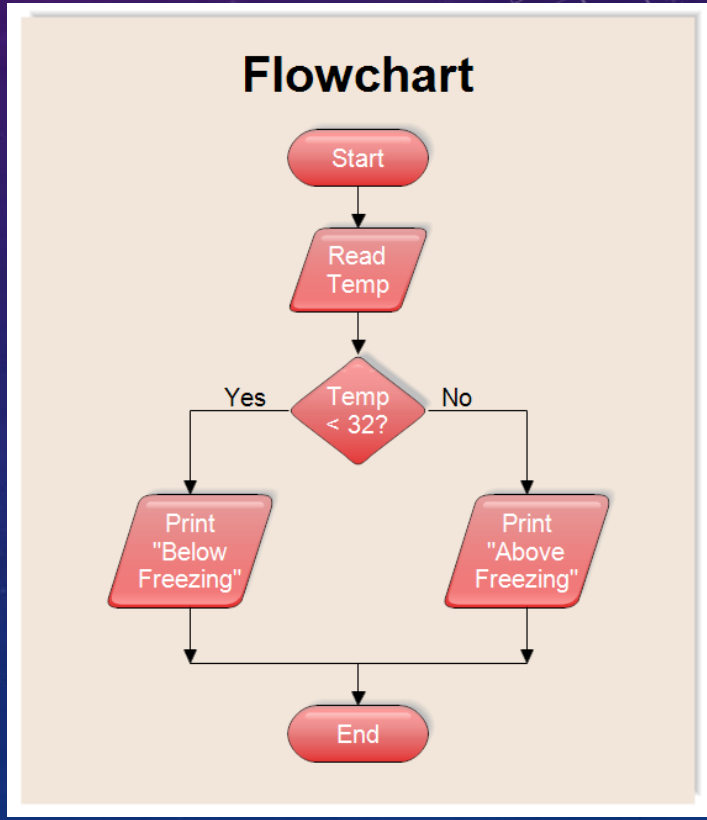
- A precise set of instructions which specifies exactly how a task is done
- Instructions must (nearly) always be carried out in correct sequence
- They are **always** language independent
- May often be changed *slightly* without generating problems – beware, computers do EXACTLY as they are TOLD (not mind readers)



FLOWCHARTING

Some rules before flowcharting

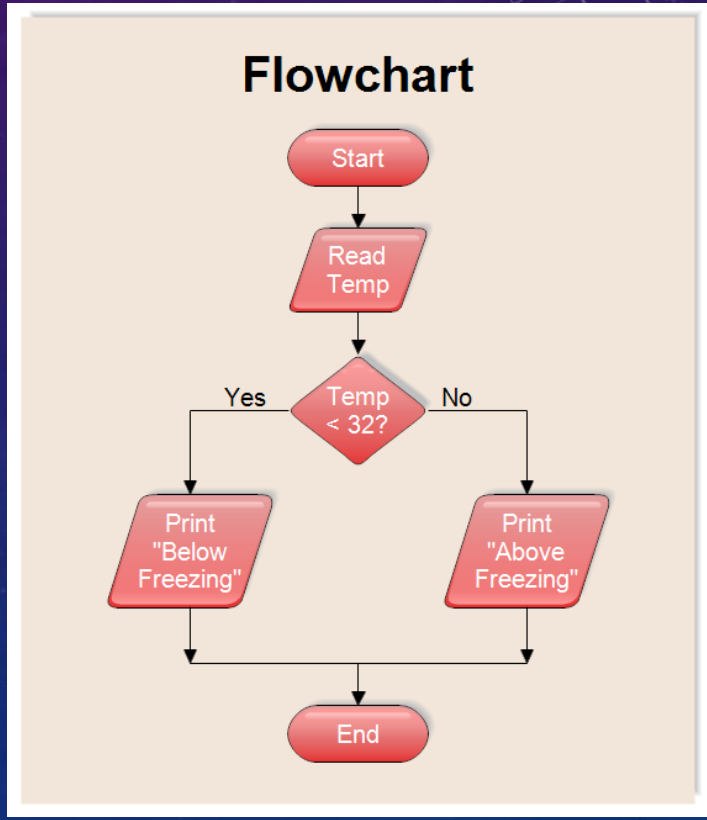
- Fully understand the problem
- Be clear what information you have and what information you are trying to discover
- Pay attention to units and conversions
- Draw a picture/sketch if possible to get problem clear in your head
- Work backwards sometimes – what is the last step required to get the required answer?
- look for repeated operations (iteration)



FLOWCHARTING

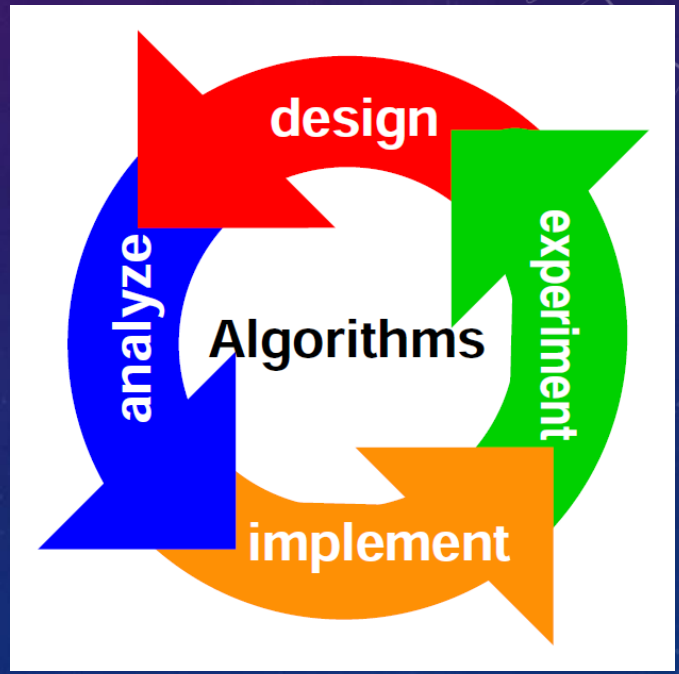
Some rules before flowcharting

- Fully understand the problem
- Be clear what information you have and what information you are trying to discover
- Pay attention to units and conversions
- Draw a picture/sketch if possible to get problem clear in your head
- Work backwards sometimes – what is the last step required to get the required answer?
- look for repeated operations (iteration)



DEVELOPING ALGORITHMS

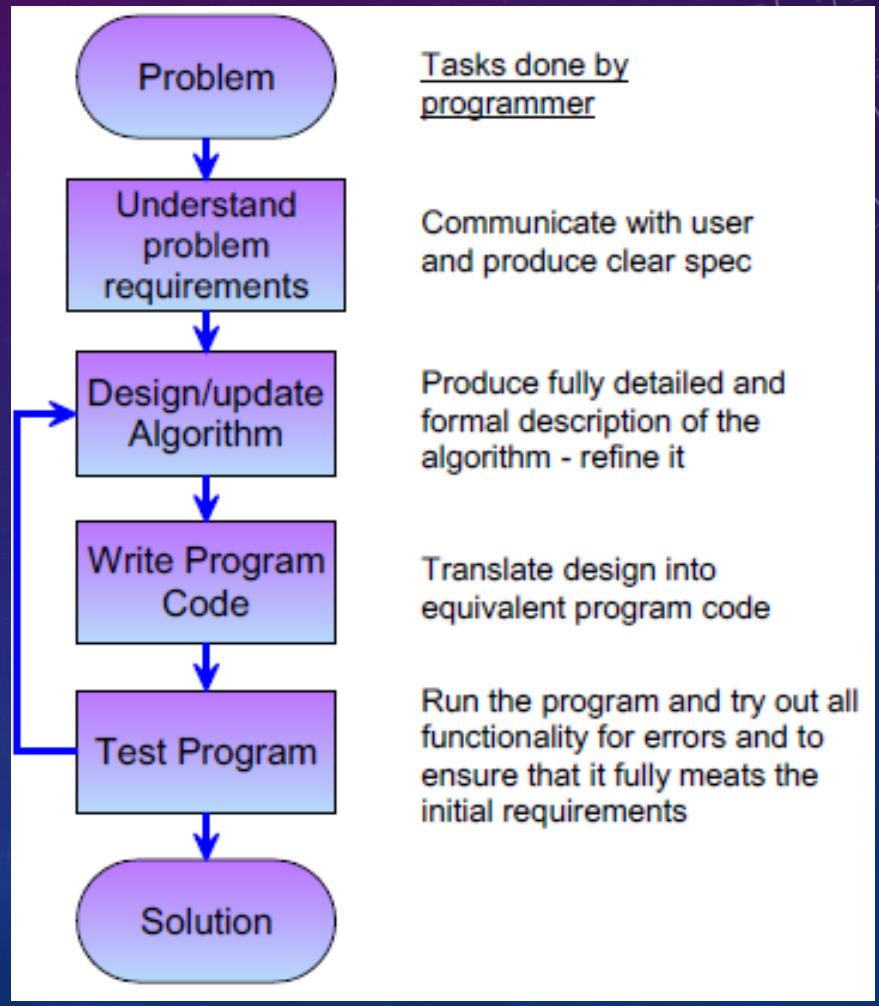
- Develop one step at a time (stepwise refinement) to produce a fully detailed and complete design
- Precise, rigorous, complete, algorithm development is essential for easy programming and fewer bugs
- Fully complete each step/stage before moving on to the next; break solution/code into small sections
- Never make changes without clearly knowing why or you simply introduce more problems!!
- If mistakes are found go back and correct from earlier stage of design (algorithm representation)



DEVELOPING ALGORITHMS

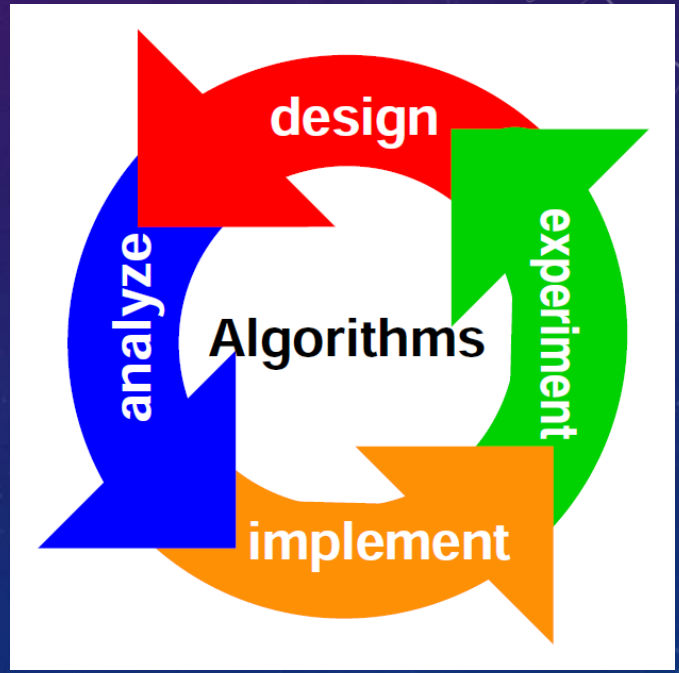
Development cycle

- Coding take up only 20% of the task
- 80% is understanding planning and thinking before writing any code



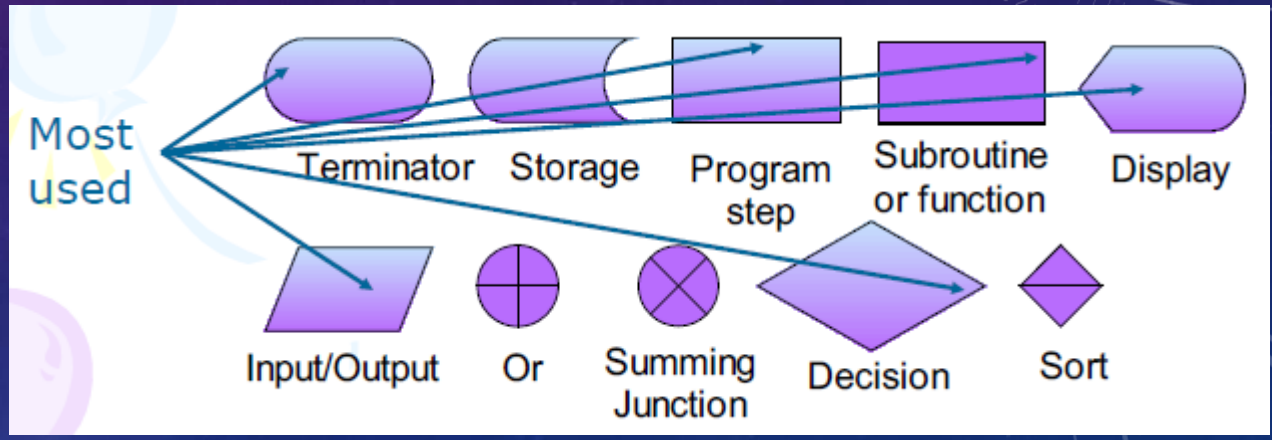
REPRESENTING ALGORITHMS

- Flowchart: clear graphical representation of Algorithm
- Pseudocode: *Structured English* using '*English-Like*' words to describe algorithm
- Flowchart much better for larger designs – but need to break large tasks down
- Goal is to produce stepwise refined *language-independent,* ***clear*** process description



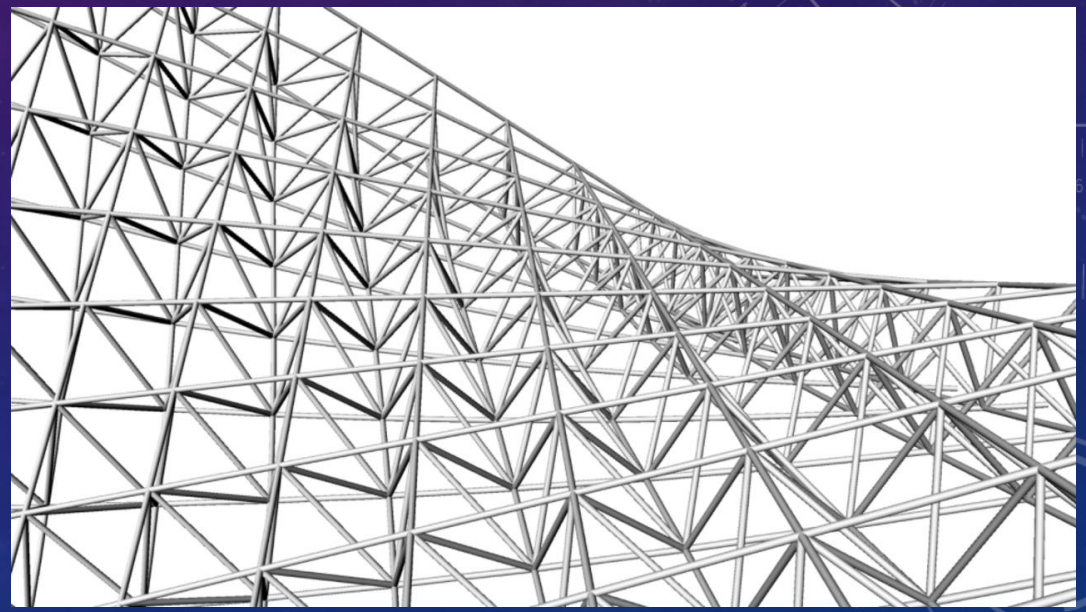
FLOWCHART SYMBOLS

- Symbols representing tasks joined with arrows indicating flow direction
- Computer generation or pencil on paper

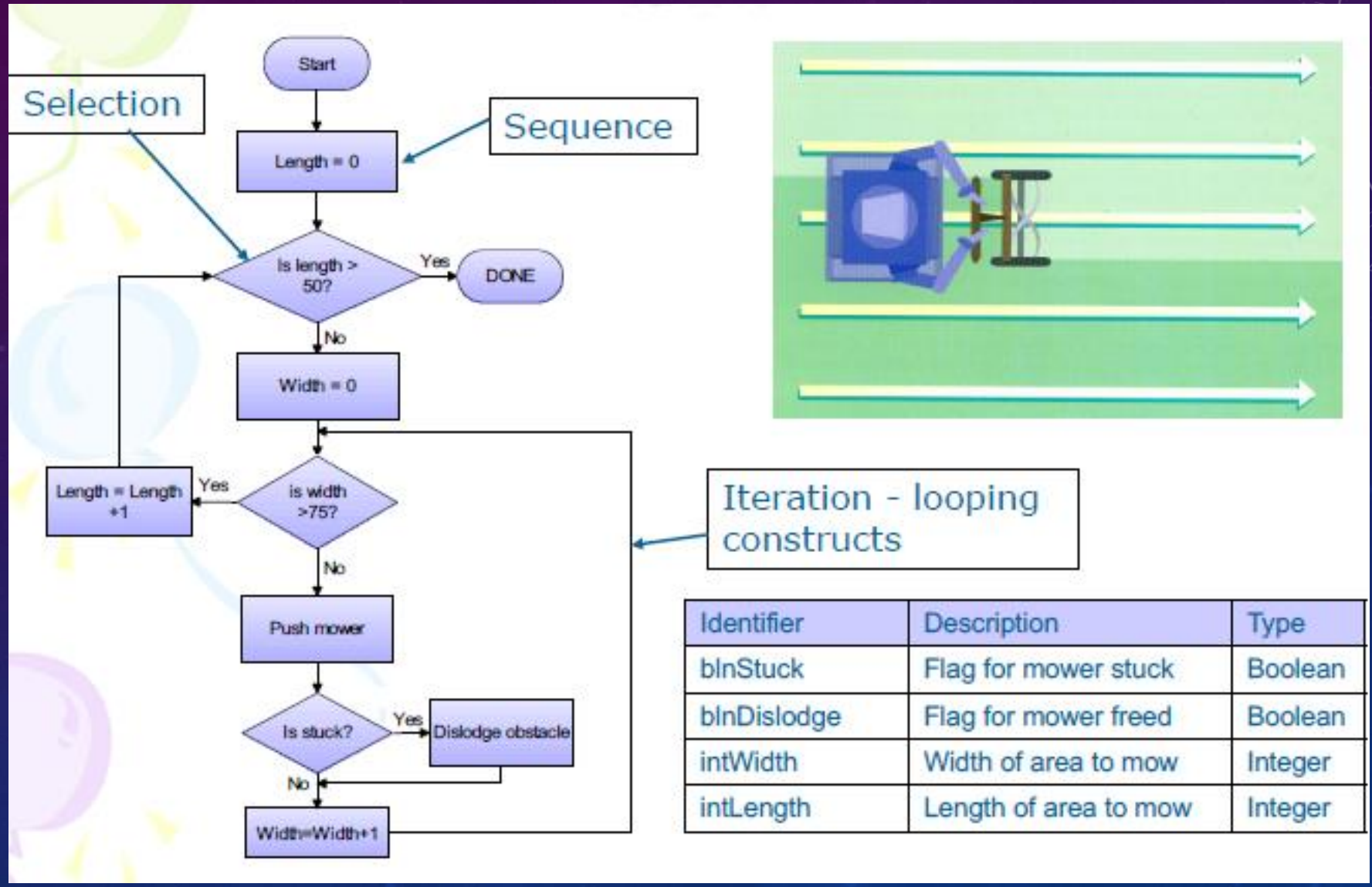


MODULARITY & STRUCTURE

- Modularity is a VERY important technique
- Build algorithm up from small sections: easy flowcharting, programming, testing, and debugging
- Split the problem up into small sections and program each as a module (*procedures - future lecture)*
- There are three basic programming structures available for use in our modules:
  - Sequence - Do this, then this, then this, then….
  - Selection - Do this, or this, or this and this, or….
  - Iteration - Do this same thing over and over again….



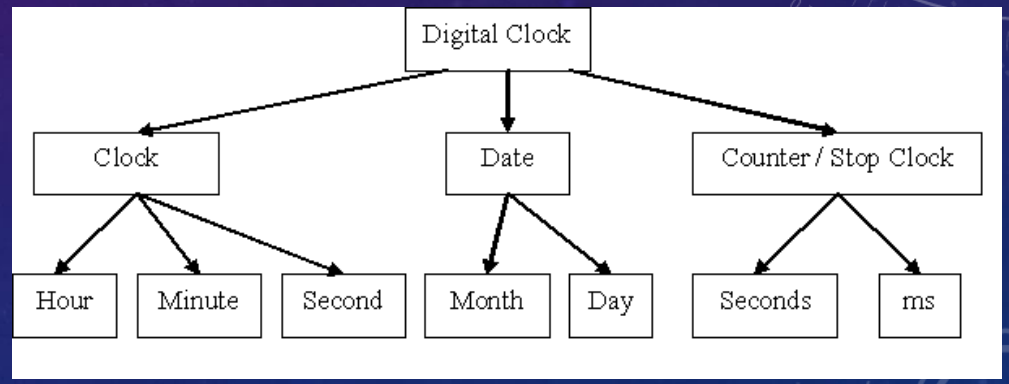
FLOWCHART EXAMPLE



Identifier	Description	Type
blnStuck	Flag for mower stuck	Boolean
blnDislodge	Flag for mower freed	Boolean
intWidth	Width of area to mow	Integer
intLength	Length of area to mow	Integer

TOP-DOWN DESIGN

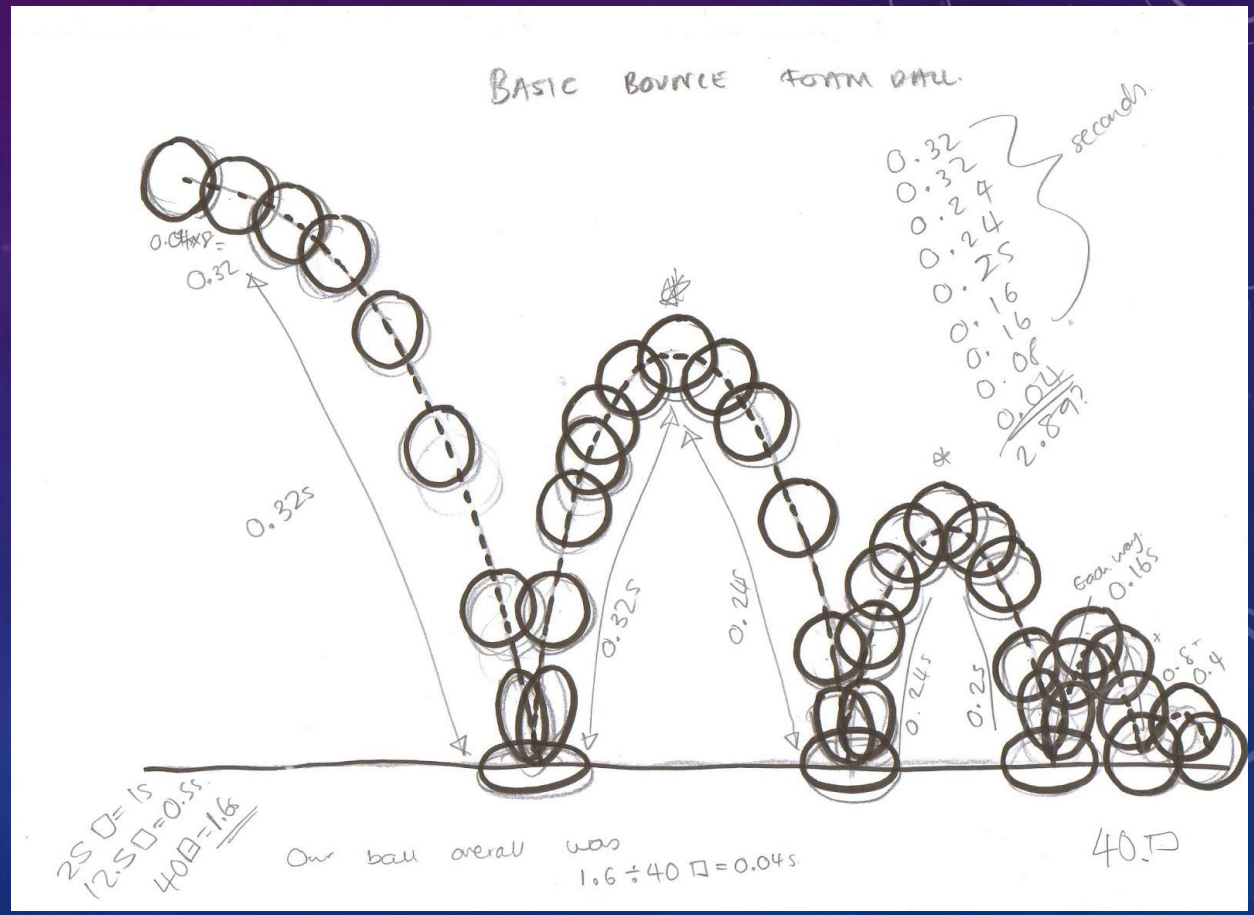
- Must fully understanding nature and complexity of the problem first
- Initially describe algorithm's functionality at high level of abstraction using basic, broad, steps
- Refine each step individually by adding increasing levels of detail; *stepwise refinement*
- A fully refined design is detailed to a level that requires a **simple** step to translate into ANY high-level programming language



Good example of a top-down design

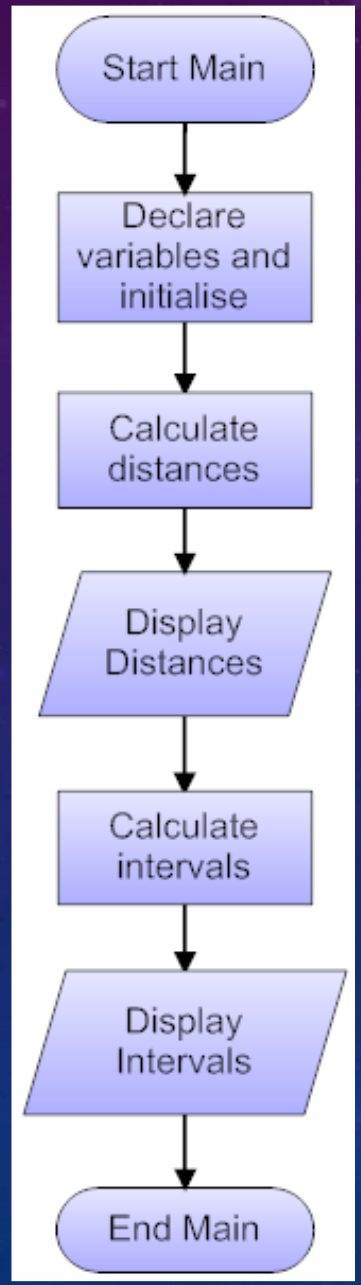
TOP-DOWN EXAMPLE

- Problem: How far does a ball fall in 1, 2, 3, and 4s if dropped? Neglect all forces except gravity. How far does it fall in each 1s interval?
- Known information: Stationary at T=0, accelerates 9.81m/s2, no other forces
- Need to know: distance travelled each second
- a = 9.81 m/s2
- Integration: v= ∫ 9.81m/s2 dt = 9.81m/s
- Integration: d= ∫ 9.81m/s dt = 0.5*9.81*t2 m



TOP LEVEL DESIGN

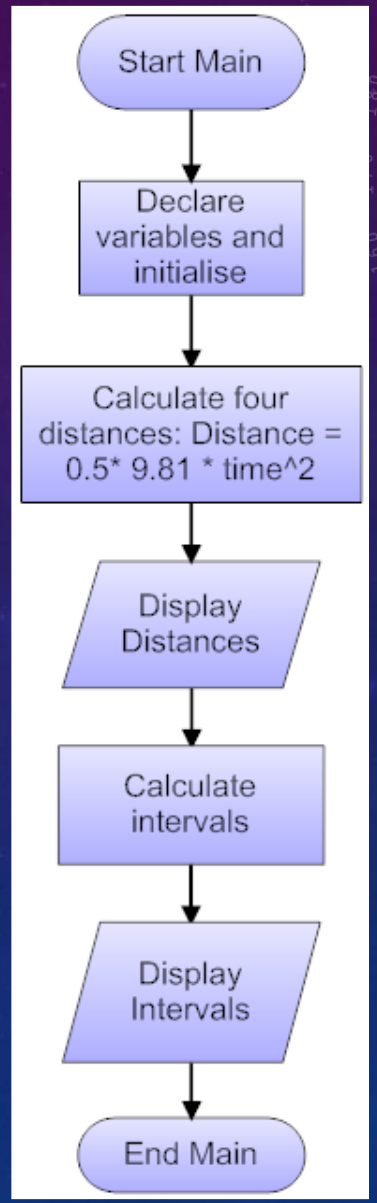
Describe algorithm with a very basic set of steps



TOP LEVEL DESIGN

Stepwise refine 1st time

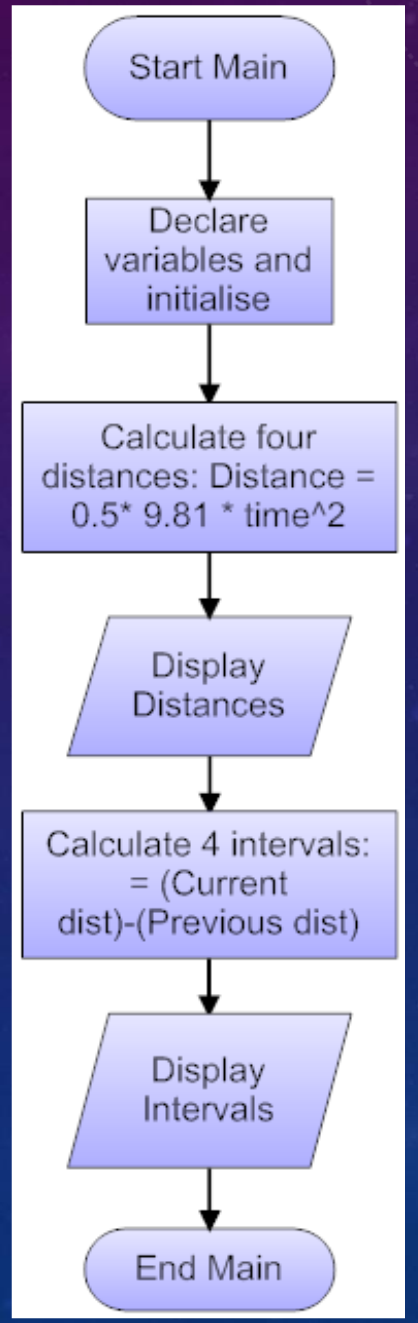
- Increase level of detail in distances Section
- Closer to a workable design realisation



TOP LEVEL DESIGN

Stepwise refine 2nd time

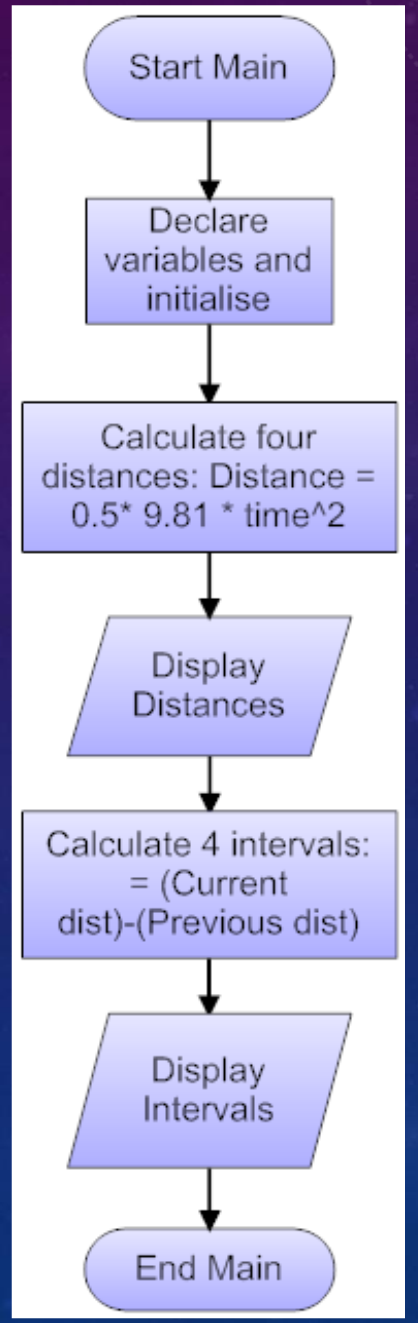
- Increase level of detail in interval Section



TOP LEVEL DESIGN

Final Refinement

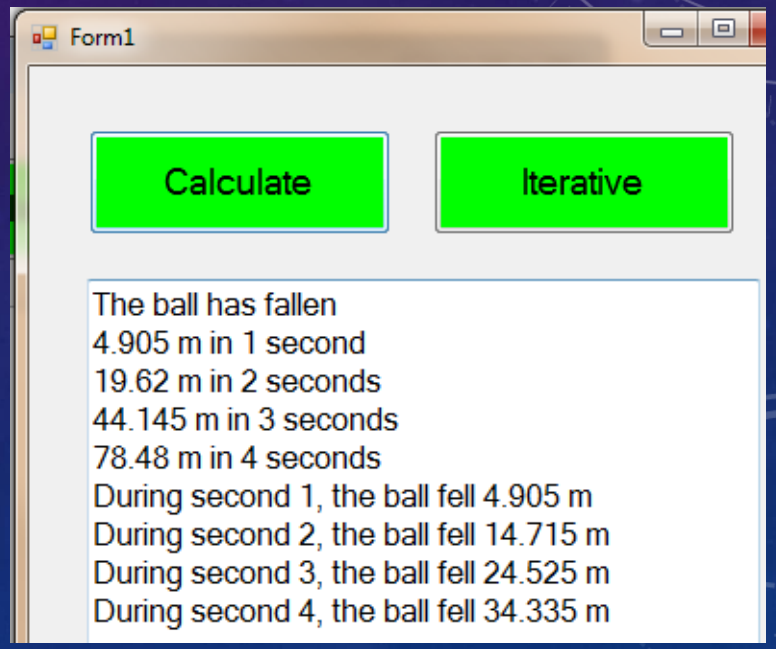
- This problem is suited to an iterative solution
- This is a workable solution that is easily programmed
- NOTE language independent!!



TOP LEVEL DESIGN

Solution for 2nd refinement

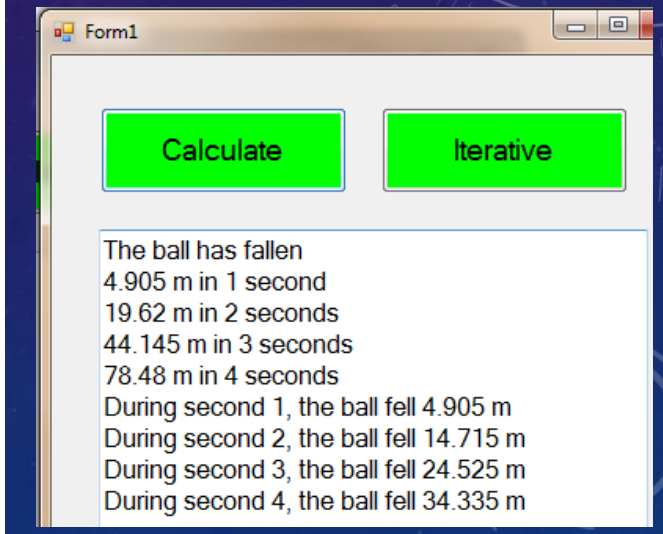
```
Private Sub Calculate_fall(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim dblDistance_1, dblDistance_2, dblDistance_3, dblDistance_4 As Double

    Const gravity = 9.81    'm/s^2

    dblDistance_1 = 0.5 * gravity * 1 ^ 2
    dblDistance_2 = 0.5 * gravity * 2 ^ 2
    dblDistance_3 = 0.5 * gravity * 3 ^ 2
    dblDistance_4 = 0.5 * gravity * 4 ^ 2

    'display running distance
    txtDisplayResult.Text = "The ball has fallen" & vbCrLf & dblDistance_1 & " m in 1 second" & vbCrLf _
    & dblDistance_2 & " m in 2 seconds" & vbCrLf & dblDistance_3 & " m in 3 seconds " & vbCrLf _
    & dblDistance_4 & " m in 4 seconds" & vbCrLf

    'display individual distances
    txtDisplayResult.Text = txtDisplayResult.Text & "During second 1, the ball fell " & dblDistance_1 _
    & " m" & vbCrLf & "During second 2, the ball fell " & (dblDistance_2 - dblDistance_1) & " m" _
    & vbCrLf & "During second 3, the ball fell " & (dblDistance_3 - dblDistance_2) & " m" & vbCrLf _
    & "During second 4, the ball fell " & (dblDistance_4 - dblDistance_3) & " m" & vbCrLf

End Sub
```



TOP LEVEL DESIGN

Solution for final refinement

```
Private Sub Iterative_Calculation(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim dblDistance(0 To 4) As Double
    Dim intTime As Integer
    Const Gravity = 9.81 'm/s^2

    txtDisplayResult.Text = "The ball has fallen" & vbCrLf

    'display running distance
    For intTime = 1 To 4
        dblDistance(intTime) = 1 / 2 * Gravity * intTime ^ 2
        txtDisplayResult.Text = txtDisplayResult.Text & dblDistance(intTime) & " m in " & intTime & " seconds" & vbCrLf
    Next

    'display individual distances
    For intTime = 1 To 4
        txtDisplayResult.Text = txtDisplayResult.Text & "During second " & intTime & ", the ball fell " & _
        (dblDistance(intTime) - dblDistance(intTime - 1)) & " m " & vbCrLf
    Next

End Sub
```



REAL WORLD ALGORITHMS

Algorithms are not just for programming

Changing a car tyre:

I. *Apply handbrake*
II. Get jack and spare tyre from boot
III. Loosen wheel nuts
IV. Insert jack and raise car
V. Remove nuts and wheel
VI. Put on spare tyre and nuts
VII. Lower car and remove jack
VIII. Tighten wheel nuts
IX. Put Jack and punctured tyre away

REAL WORLD ALGORITHMS

What about changes to an algorithm?

- Omit steps or change order in wheel change algorithm? Disaster!

- *Change steps 4 & 5 around! Miss out 6!*

- Computers are really stupid, not intelligent

- They do EXACTLY as they are told – not what the programmer intended!

- Get algorithm RIGHT before coding

- The work is algorithm design, not the coding

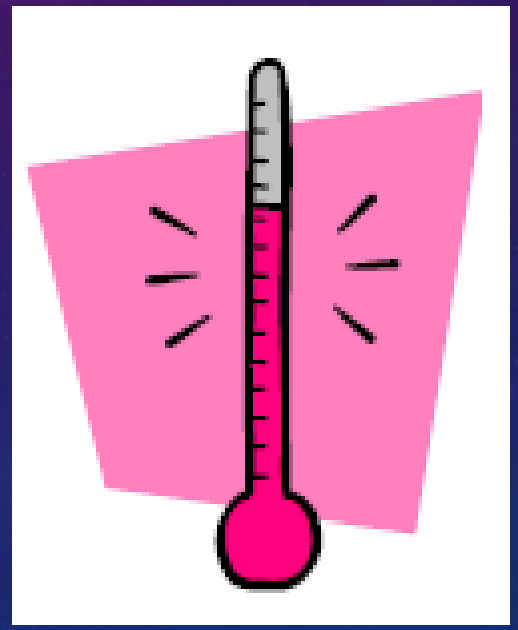
REAL WORLD ALGORITHMS

Example top down design

Understand problem

- *How to convert ˚C to ˚F?*
- *From research ((Centigrade*9)/5)+32*

- Consider interface design at this point
- Care during algorithm development: determine order of execution of steps

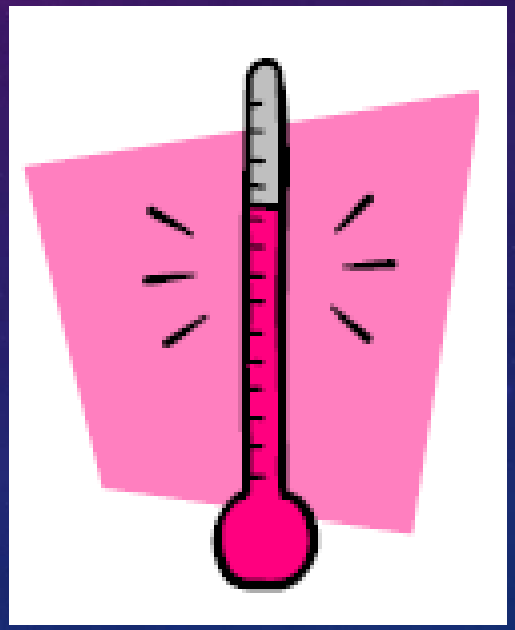


REAL WORLD ALGORITHMS

Top-Level design (pseudocode)

1 Read in centigrade value
2 Calculate Fahrenheit value
3 Display Fahrenheit value

Now stepwise refine the algorithm:
- *What **type** of values will be read in/out?*
- *How will we store them? – create a data table*
- *How do we perform the conversion?*



REAL WORLD ALGORITHMS

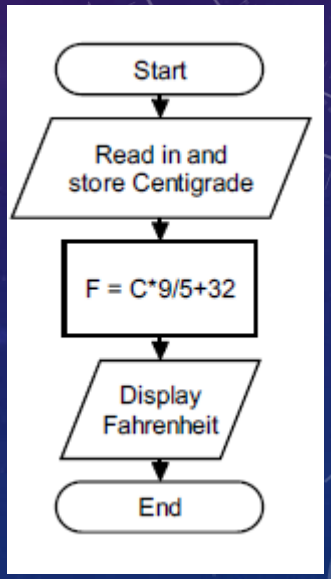
Stepwise refined design

1 Read in centigrade value
2.1 Set Fahrenheit to (C*9/5 + 32)
3 Display Fahrenheit value

Now stepwise refine the algorithm:
- *What **type** of values will be read in/out?*
- *How will we store them? – create a data table*
- *How do we perform the conversion?*

Data table

Identifier	Description	Type
Centigrade	Initial temp in °C	Single
Fahrenheit	Equiv temp in °F	Single



EXERCISE (10 – 15 MINS)

- Design a Centigrade to Fahrenheit convertor
- There is no need to design it vice-versa
- Popular exam question to test competency
- Formula:
  Fahrenheit = Centigrade * 9/5+32