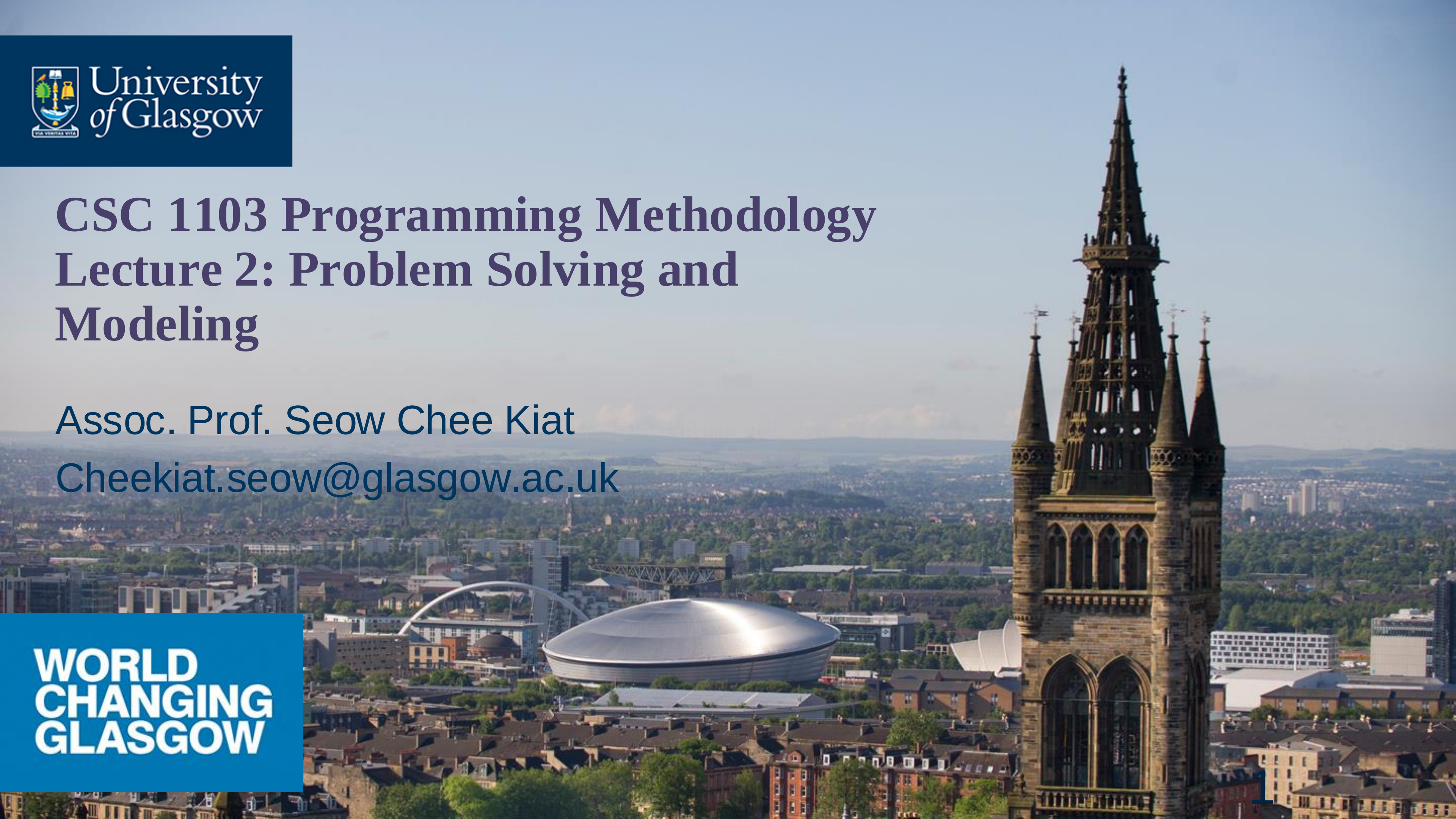
University of Glasgow

CSC 1103 Programming Methodology Lecture 2: Problem Solving and Modeling

Assoc. Prof. Seow Chee Kiat
Cheekiat.seow@glasgow.ac.uk

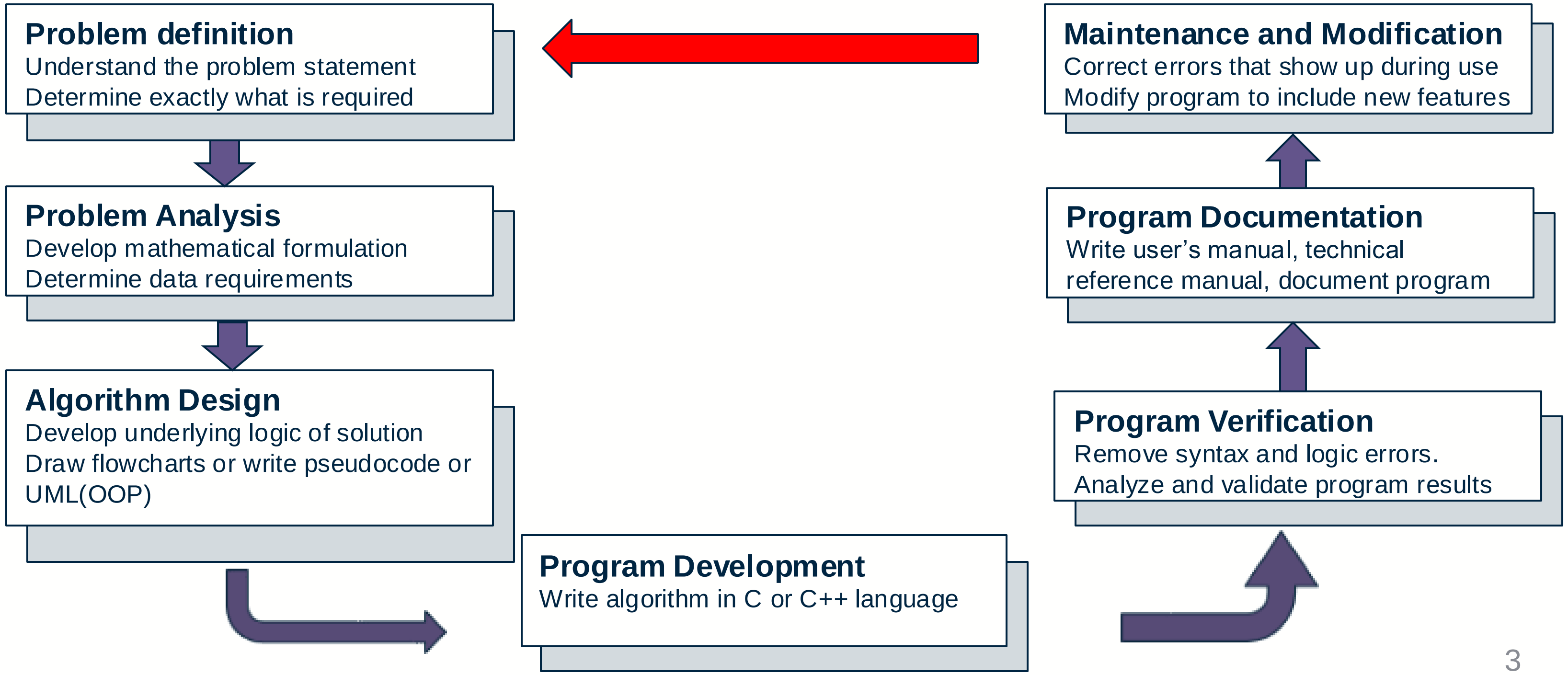
WORLD CHANGING GLASGOW



Problem Solving

- Before programming coding is performed, given a problem statement, how should we approach to solve it? Traditionally, 7 step approach for software engineering
    - Problem Definition
    - Problem Analysis
    - Algorithm Design
    - Program development
    - Program Verification
    - Documentation
    - Maintenance and Modification

Problem Solving



Problem definition
Understand the problem statement
Determine exactly what is required

Problem Analysis
Develop mathematical formulation
Determine data requirements

Algorithm Design
Develop underlying logic of solution
Draw flowcharts or write pseudocode or UML(OOP)

Program Development
Write algorithm in C or C++ language

Program Verification
Remove syntax and logic errors.
Analyze and validate program results

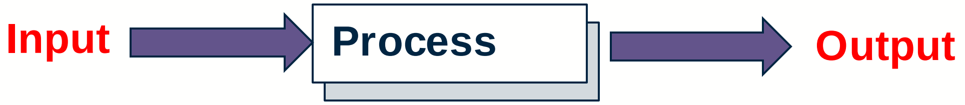
Program Documentation
Write user’s manual, technical reference manual, document program

Maintenance and Modification
Correct errors that show up during use
Modify program to include new features

University of Glasgow | School of Computing Science

Problem Solving

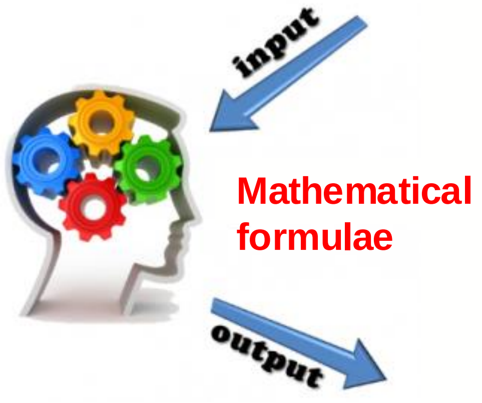
- **Problem Definition**
  - Define the problem.
  - Clear understanding of the problem to be solved
  - Problem statement should be clearly examined to determine what information is given (the input data) and what need to be found, what results should be computed and display (the output)
  - If initial description of the problem is not clear or vague and imprecise, reformulate the original statement so that it is clear and concise.
    - Definite input data
    - Concise output to be created



University of Glasgow School of Computing Science

Problem Solving

- **Problem Analysis**
  - Develop a mathematical formulation once the input and output requirement is clear.
  - Requires a clear understanding of the underlying concepts and principle inherent to the problem by assembling the information (input and output) needed
  - The relationship between the input and output variables are obtained
    - Relationship expressed in terms of mathematical equations
    - Assumption on the mathematical modelling and equations assumption should be made and stated



Problem Solving

- **Algorithm Design**
  - Formulate the algorithm
    - How to implement the mathematical formulation regarding to the input and output that is derived in Problem Analysis step
  - Methodology in algorithm design
    - Top down (procedural programming like C),
    - bottom up design (object oriented programming like C++)
    - Computational Thinking (CT)

University of Glasgow | School of Computing Science

Problem Solving

- E.g. Simple Problem statement : Find the square root of a non complex number
  - **Problem definition**: Input : A non negative non complex number

    Output : The value after the square root of the input

    Problem Analysis : Mathematical function that relates to input and output : $output = \sqrt[2]{input}$

    Constraint : Non negative & non complex number check

    Algorithms : 1. Read in the input number

    2. Check the sign of number for non complex and non negative

    3. If the input number is complex or negative, throw an error message, else go to step 4

    4. Compute square root of the input number for the output

    5. Print the input number and the output

University of Glasgow | School of Computing Science

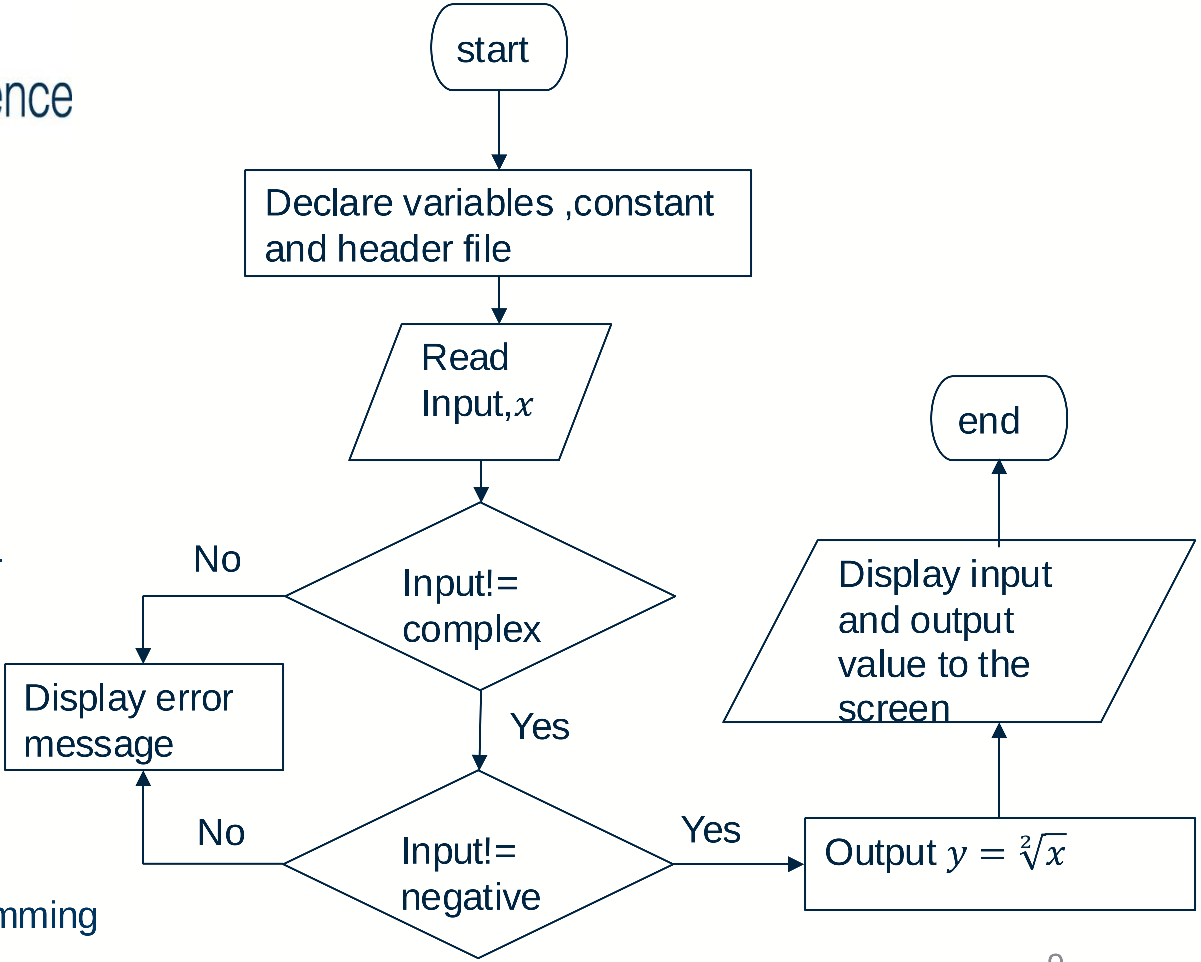
Problem Solving

- Problem statement : Finding the least square solution of a time varying Wi-Fi signal input $\mathbf{x}=[x_1 \quad \dots \quad x_n]$ that reach the handphone having output $\mathbf{y}=[y \quad \dots \quad y_n]$ assuming the channel is a quasi stationary process $\mathbf{h}=[h_1 \quad \dots \quad h_n]$ with guassian noise
  - How to write the algorithm if the solution is $\mathbf{x}=(\mathbf{h}^T\mathbf{h})^{-1}\mathbf{h}^T\mathbf{y}$?
- For complex problem statement, step by step English descriptions such as in previous example are inadequate for describing algorithms. Two common methods/tools for describing algorithm
  - Flowchart
    - Visual representation of data flow of the algorithm and sequence of operations through interconnected standard unique symbols
  - Pseudocode
    - Highly Informal language to outline structure, logic structure of the and sequence of operations of the algorithm

University of Glasgow | School of Computing Science

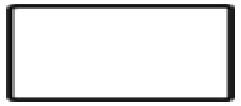
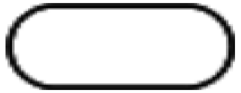
Problem Solving

- Flowchart
  - composed of a set of standard symbols, each of which is unique in shape and represents a particular operation
  - Symbols are Connected by straight lines arrow called flowlines.
  - Independent of programming languages



Problem Solving

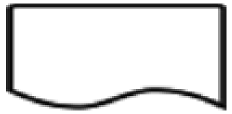
- Flowchart

Flowchart Symbol	Name	Description
	Process symbol	Also known as an "Action Symbol," this shape represents a process, action, or function. It's the most widely-used symbol in flowcharting.
	Start/End symbol	Also known as the "Terminator Symbol," this symbol represents the start points, end points, and potential outcomes of a path. Often contains "Start" or "End" within the shape.

University of Glasgow | School of Computing Science

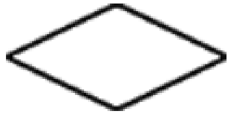
Problem Solving

- Flowchart



Document symbol

Represents the input or output of a document, specifically. Examples of and input are receiving a report, email, or order. Examples of an output using a document symbol include generating a presentation, memo, or letter.



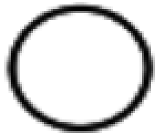
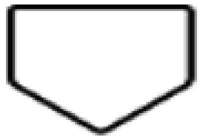
Decision symbol

Indicates a question to be answered — usually yes/no or true/false. The flowchart path may then split off into different branches depending on the answer or consequences thereafter.

University of Glasgow | School of Computing Science

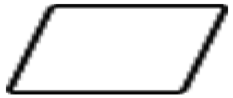
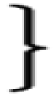
Problem Solving

- Flowchart

	Connector symbol	Usually used within more complex charts, this symbol connects separate elements across one page.
	Off-Page Connector/Link symbol	Frequently used within complex charts, this symbol connects separate elements across multiple pages with the page number usually placed on or within the shape for easy reference.

Problem Solving

- Flowchart

	Input/Output symbol	Also referred to as the "Data Symbol," this shape represents data that is available for input or output as well as representing resources used or generated. While the paper tape symbol also represents input/output, it is outdated and no longer in common use for flowchart diagramming.
	Comment/Note symbol	Placed along with context, this symbol adds needed explanation or comments within the specified range. It may be connected by a dashed line to the relevant section of the flowchart as well.

- for more flowchart symbol can refer to any flowchart application such lucidchart : https://www.lucidchart.com/pages/flowchart-symbols-meaning-explained

University of Glasgow School of Computing Science

Problem Solving

- Pseudocode
  - Consists of series of statements and instructions that when followed will solve the problem
  - Uses a mix of English, mathematical notation and a set of special words (called) keywords to describe the operation of the algorithm
  - Keywords such as BEGIN, END, READ, PRINT, IF, ELSE, ELSEIF, ENDIF, WHILE, DO, ENDWHILE AND ENDDO
  - Keywords are capitalized while processing steps and conditions are written in lowercase. A separate line is used for each distinct step of algorithm and statements between keywords are indented to clarify structure of algorithm. Computations are indicated by ←
  - Programming language independent and much preferred over flowchart as it is closer to actual program code implementation.

```
BEGIN
        Task 1
        Task 2
END
```

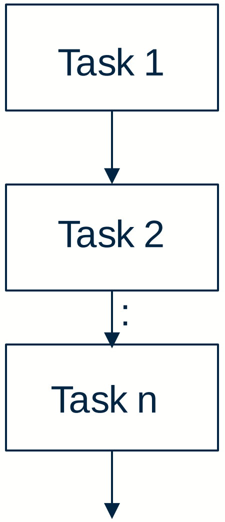
Problem Solving

- Control Structure
  - All algorithms regardless how complex is a combination of three fundamental control structure
    - Sequence Structure
      - One operation or group of operations is performed after the other sequentially . All programs are considered so have been sequential
    - Selection Structure
      - Test a condition to determine which steps are to be performed next. Different operations are being selected based on the tested condition is true or false
    - Repetition
      - A series of steps is repeated such as in conditional loop and counting loop

School of
Computing Science

Problem Solving

- Sequence Structure



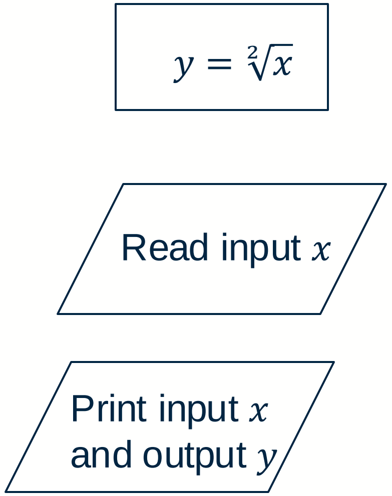
Flowchart

```
BEGIN
        Task 1
        Task 2
          :
        Task n
END
```

Pseudocode

Problem Solving

- Sequence Structure e.g.



```
$y \leftarrow \sqrt[2]{x}$
READ input $x$
PRINT input $x$ and output $y$
```

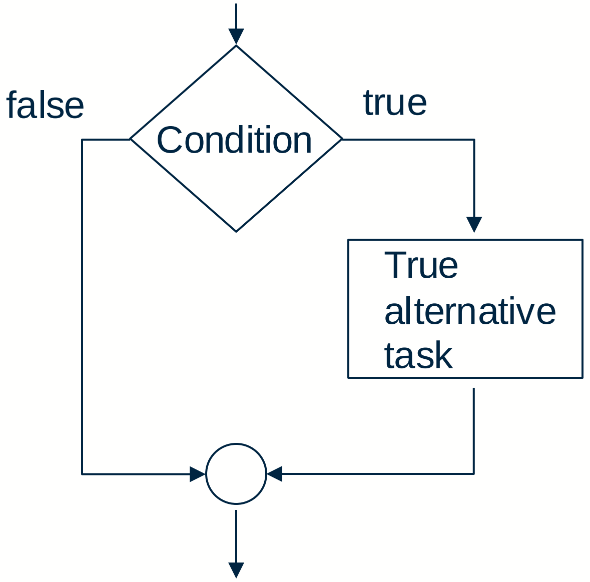
Flowchart

Pseudocode

University of Glasgow School of Computing Science

Problem Solving

- Selection Structure : IF Structure



Flowchart

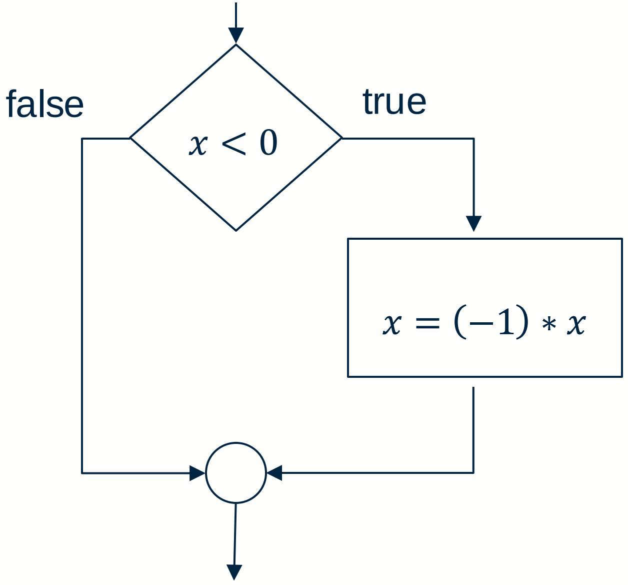
```
IF condition
          True alternative task
ENDIF
```

Pseudocode

University of Glasgow | School of Computing Science

Problem Solving

- Selection Structure : IF Structure e.g. Obtain the absolute value of a data value



Flowchart

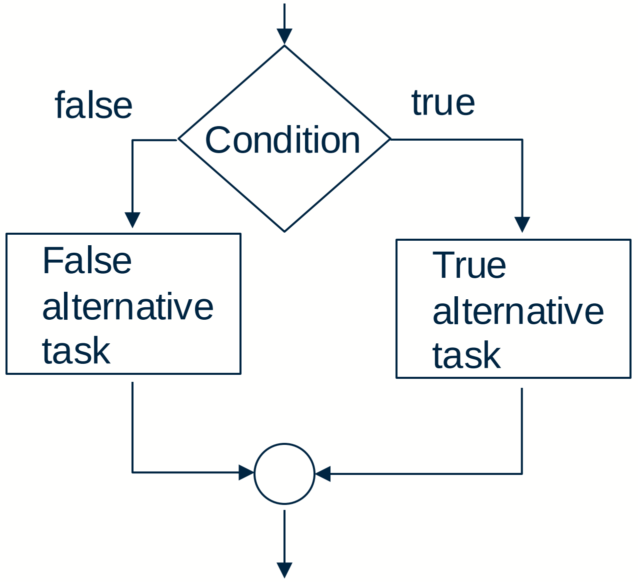
```
IF $x < 0$
        $x \leftarrow (-1) * x$
ENDIF
```

Pseudocode

University of Glasgow School of Computing Science

Problem Solving

- Selection Structure : IF-ELSE Structure



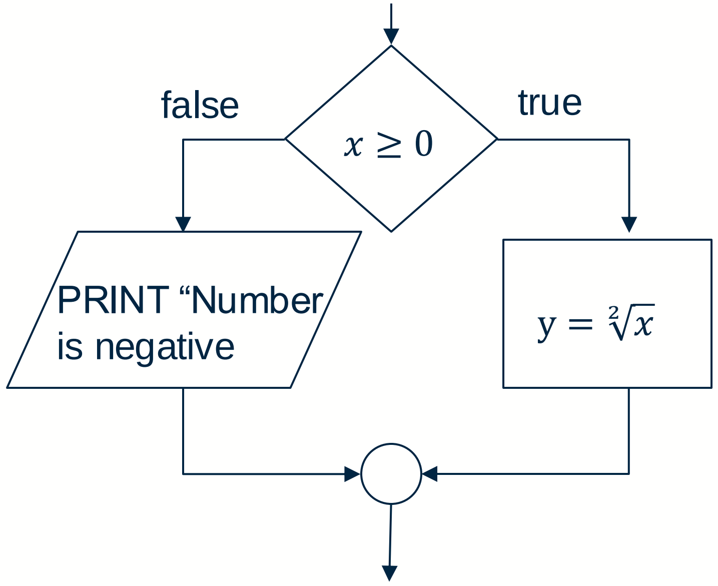
Flowchart

```
IF condition
        True alternative task
ELSE
        False alternative task
ENDIF
```

Pseudocode

Problem Solving

- Selection Structure IF-ELSE Structure e.g. Obtain the square root of a data value



Flowchart

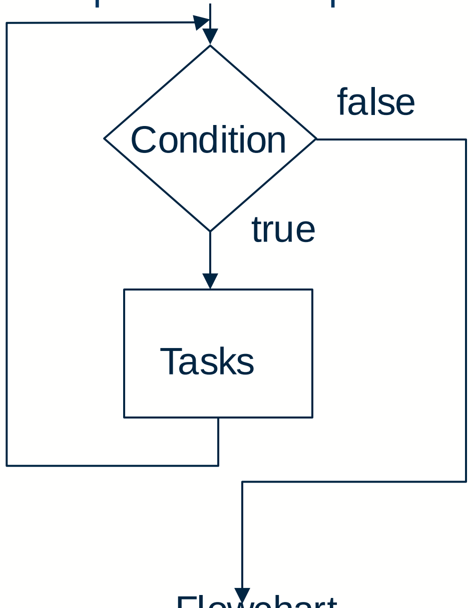
```
IF $x \geq 0$
        y← $\sqrt[2]{x}$
ELSE
        PRINT "Number is negative
ENDIF
```

Pseudocode

School of Computing Science

Problem Solving

- Repetition or Loop Structure :WHILE Loop



```
WHILE condition
        Tasks
ENDWHILE
```

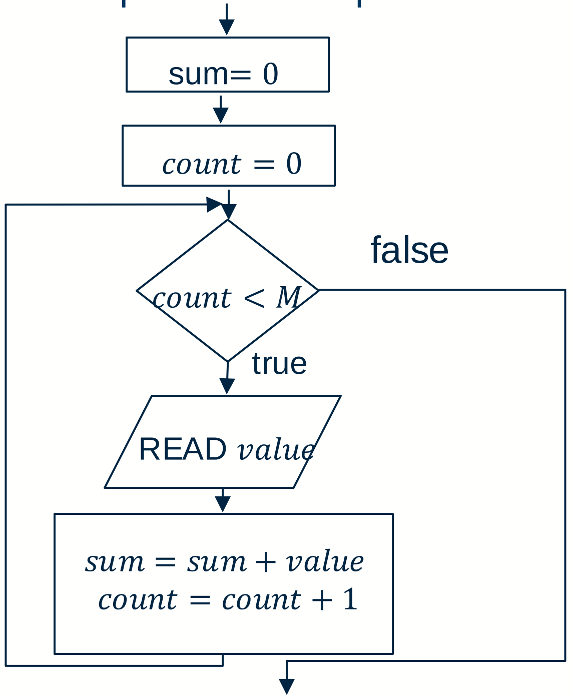
Flowchart

Pseudocode

University of Glasgow School of Computing Science

Problem Solving

- Repetition or Loop Structure :WHILE Loop e.g. Obtain the sum of M number



Flowchart

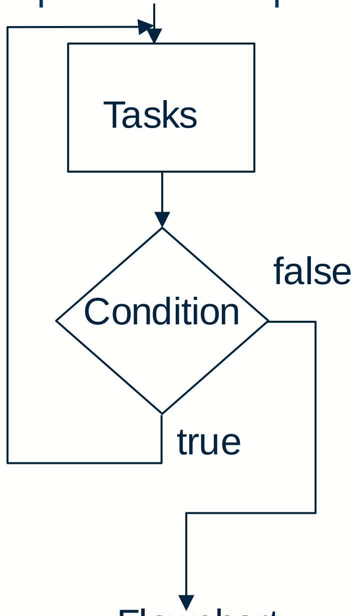
```
$count \leftarrow 0$
$sum \leftarrow 0$
WHILE $count < M$
    READ $value$
    $sum \leftarrow sum + value$
    $count \leftarrow count + 1$
ENDWHILE
```

Pseudocode

University of Glasgow School of Computing Science

Problem Solving

- Repetition or Loop Structure :DOWHILE Loop



Flowchart

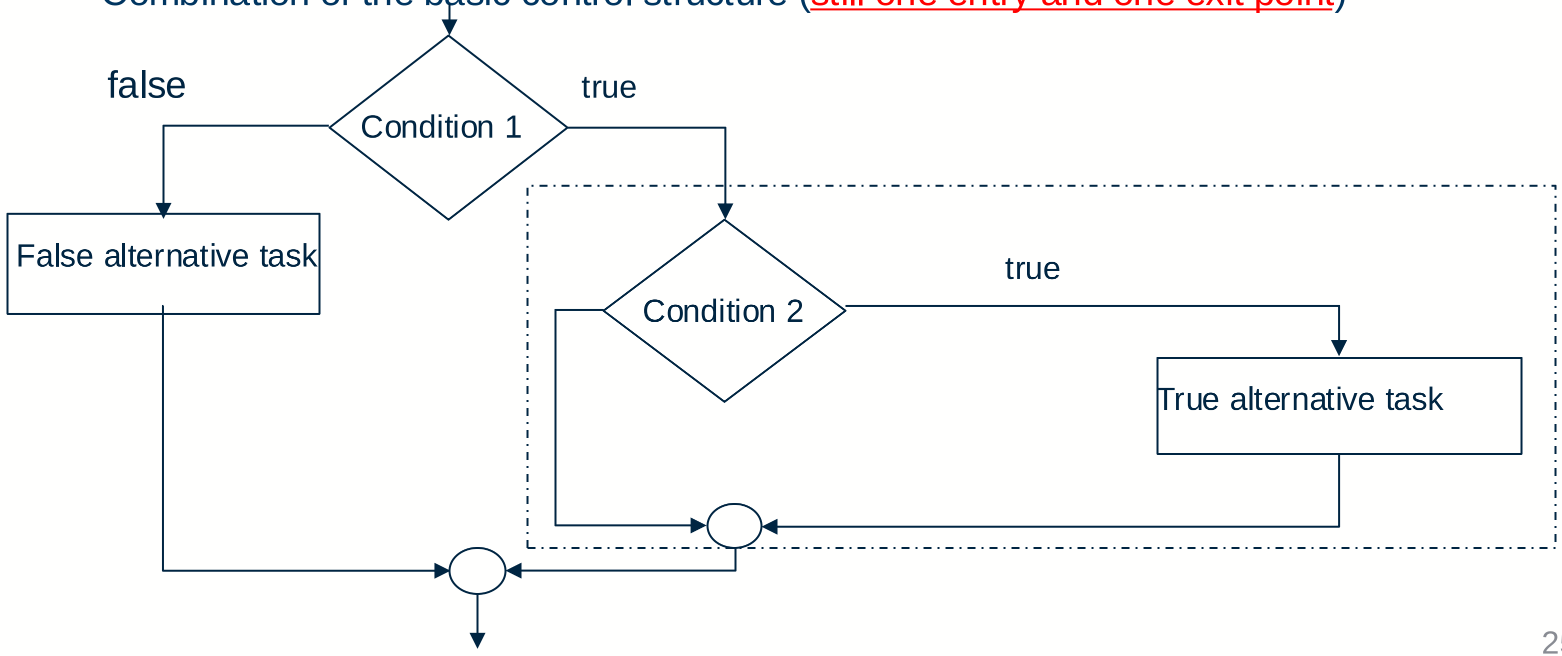
```
DO
        Tasks
WHILE condition
```

Pseudocode

University of Glasgow School of Computing Science

Problem Solving

- Combination of the basic control structure (still one entry and one exit point)



Problem Solving

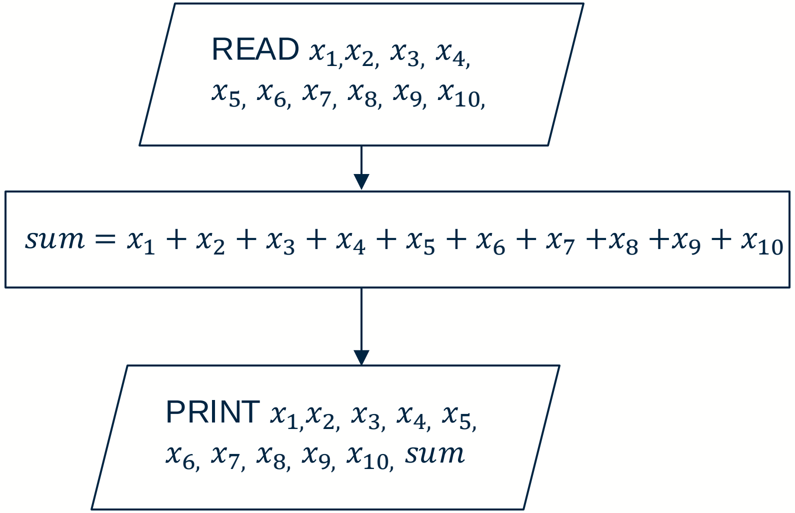
- More Example : 1)Write pseudocode and flowchart statements to describe the algorithm of computing the $\sum_{i=1}^{10} x_i$

  The pseudocode representation of the algorithm is

```
BEGIN
        READ $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}$
        $sum \leftarrow x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10}$
        PRINT $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}$ and $sum$
END
```

Problem Solving

- The flowchart representation of the algorithm


READ $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10},$
$sum = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10}$
PRINT $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, sum$

University of Glasgow | School of Computing Science

Problem Solving

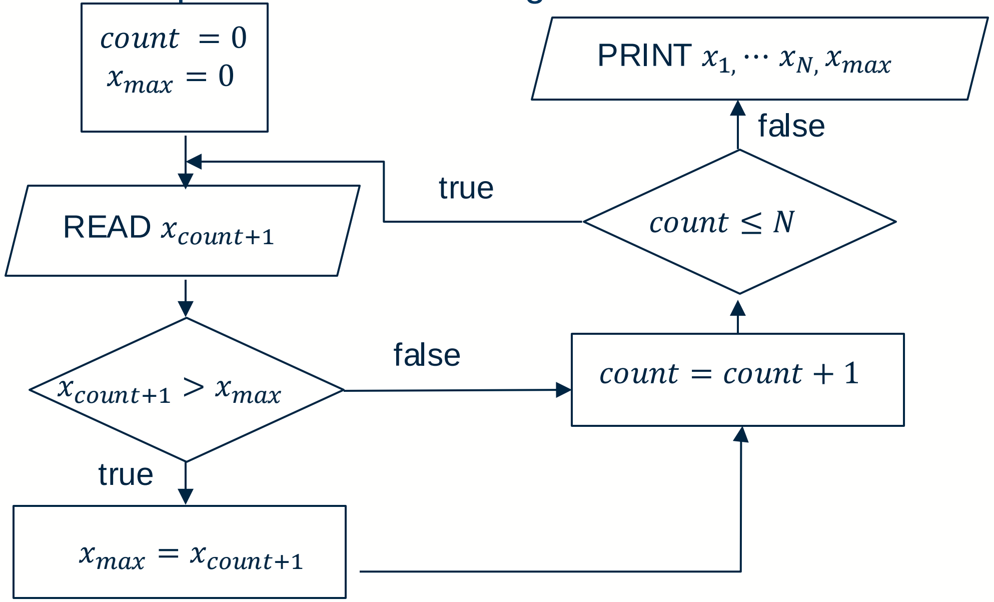
- More Example : 2)Write pseudocode and flowchart statements to describe the algorithm of finding the largest number in the set $S = \{x_1, x_2 \quad \cdots \quad x_N\}$

  The pseudocode representation of the algorithm is

```
BEGIN
        $count \leftarrow 0$
        $x_{max} \leftarrow 0$
        DO
            READ $x_{count+1}$
            IF $x_{count+1} > x_{max}$
                $x_{max} \leftarrow x_{count+1}$
            ENDIF
            $count \leftarrow count + 1$
       WHILE $count < N$
       PRINT $x_1 \cdots x_N, x_{max}$
END
```

Problem Solving

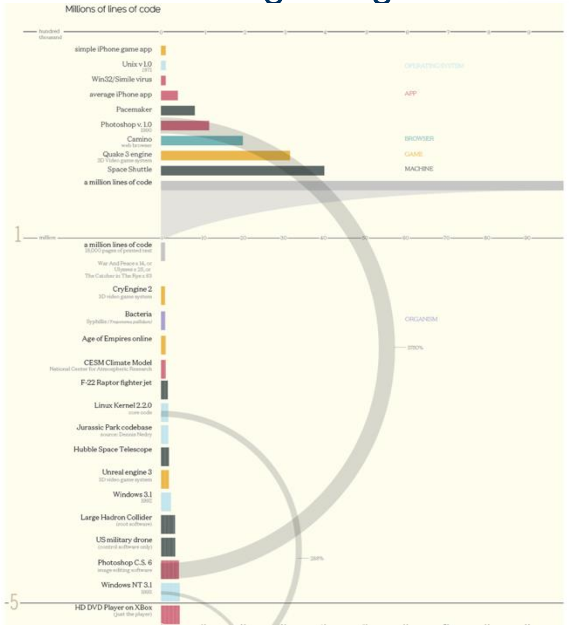
- The flowchart representation of the algorithm

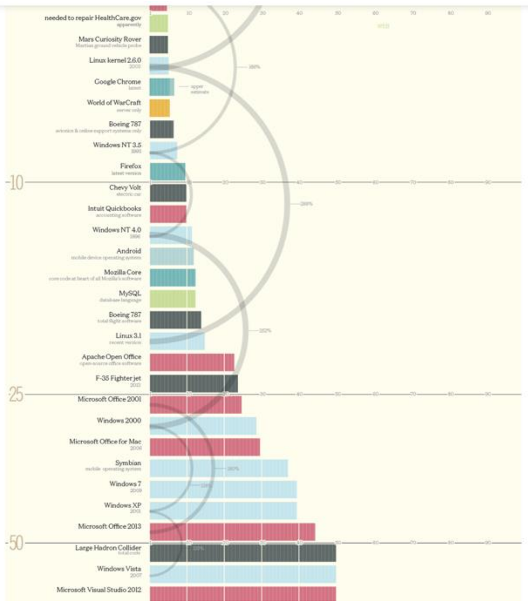


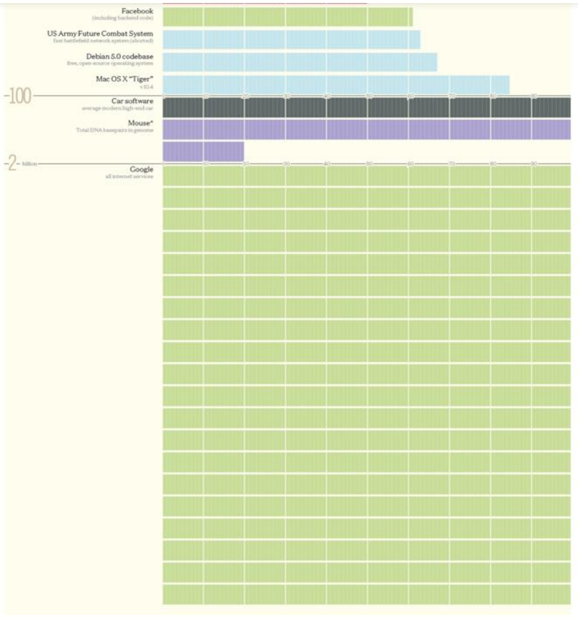
School of Computing Science

Type of Methodology in Problem Solving

- Most problem statements are usually multi-disciplined and complex in nature and need to manage large amount of data and details







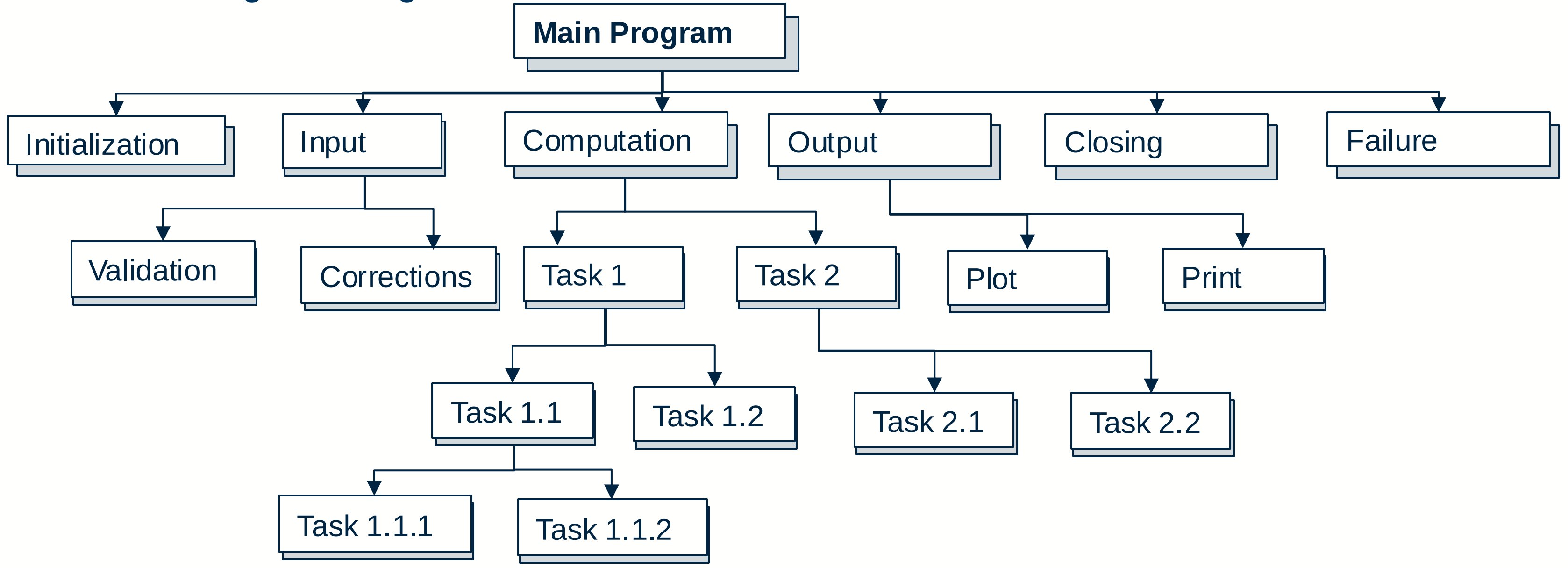
Type of Methodology in Problem Solving

- Structured Programming
    - Top down design
    - Decomposition and stepwise refinement
    - Break into many independent small modules
        - well defined function and interface
        - Change in one module should not affected other modules
        - only one entry point and exit point
    - Employ only the three fundamental control structure
    - Prohibit the use of unconditional transfer function
    - Construct for ease of readability and employability

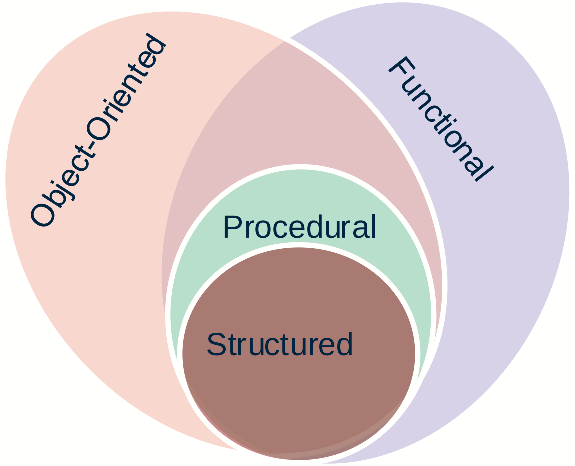
University of Glasgow | School of Computing Science

Type of Methodology in Problem Solving

- Structured Programming



Type of Methodology in Problem Solving



- Structured Programming
  - Procedural Programming (PP)
    - For C (3rd generation), programs are divided into small self-contained functions. Focus on process/logical structure.
    - Less secure as there is no way of data hiding in general
    - Less abstraction and flexibility
  - Object Oriented Programming (OOP)
    - For C++ (4th generation),programs divided into small entities called objects. Focus on data.
    - More secure as having data hiding feature
    - More abstraction and flexibility
  - Functional Programming (FP)
    - For Haskell (5th generation declarative(non-imperative)), based on mathematical function theory

University of Glasgow | School of Computing Science

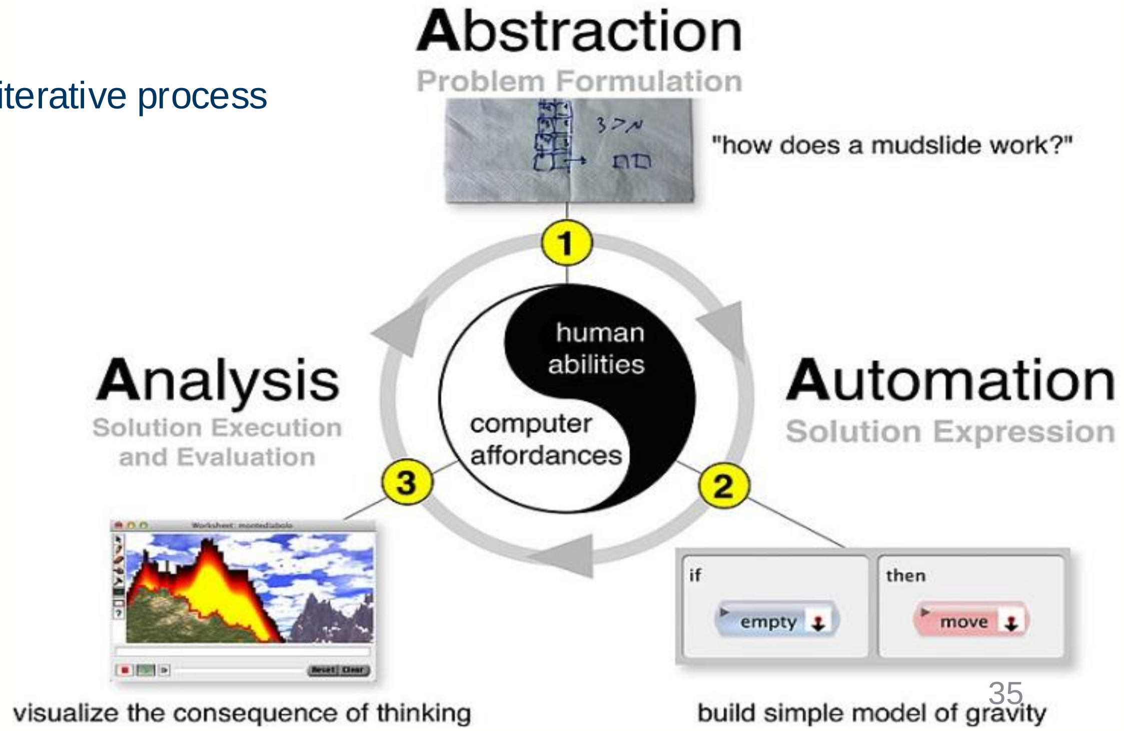
Type of Methodology in Problem Solving

- Computational Thinking (CT)
    - New “paradigm” in the methodology for problem solving
    - Four elements for strong computational thinking
        - Decomposition
            - break down a problem into subproblem
        - Pattern Recognition
            - notice similarities, difference, properties, or trends in data
        - Pattern Generalization/Abstraction
            - extract unnecessary details and generalize those that are necessary in order to define a function/concept in general
        - Algorithm Design
            - Build repeatable, step-by-step process to solve the problem

University of Glasgow | School of Computing Science

Type of Methodology in Problem Solving

- Computational Thinking (CT)
  - Another manifestation : Three A iterative process
    - Abstraction
      - Problem formulation
    - Automation
      - Solution Expression
    - Analyses
      - Solution Execution


Abstraction
Problem Formulation
"how does a mudslide work?"
1
human abilities
computer affordances
Automation
Solution Expression
2
if
empty
then
move
build simple model of gravity
Analysis
Solution Execution and Evaluation
3
visualize the consequence of thinking

University of Glasgow School of Computing Science

Type of Methodology in Problem Solving

- E.g. Apply Computational Thinking to describe the algorithm of finding the smallest and largest integer number in the set $S = \{x_1, x_2 \quad \cdots \quad x_N\}$
    - Decomposition
        - Input = N number of x variable
        - Process = a mathematical function to find the smallest number of these N variables
        - Output = print out the N variables and their smallest number
    - Pattern Recognition
        - Require the need to do comparison for
            - N variable x for both largest and smallest number
        - N variable x and count are integer

University of Glasgow | School of Computing Science

Type of Methodology in Problem Solving

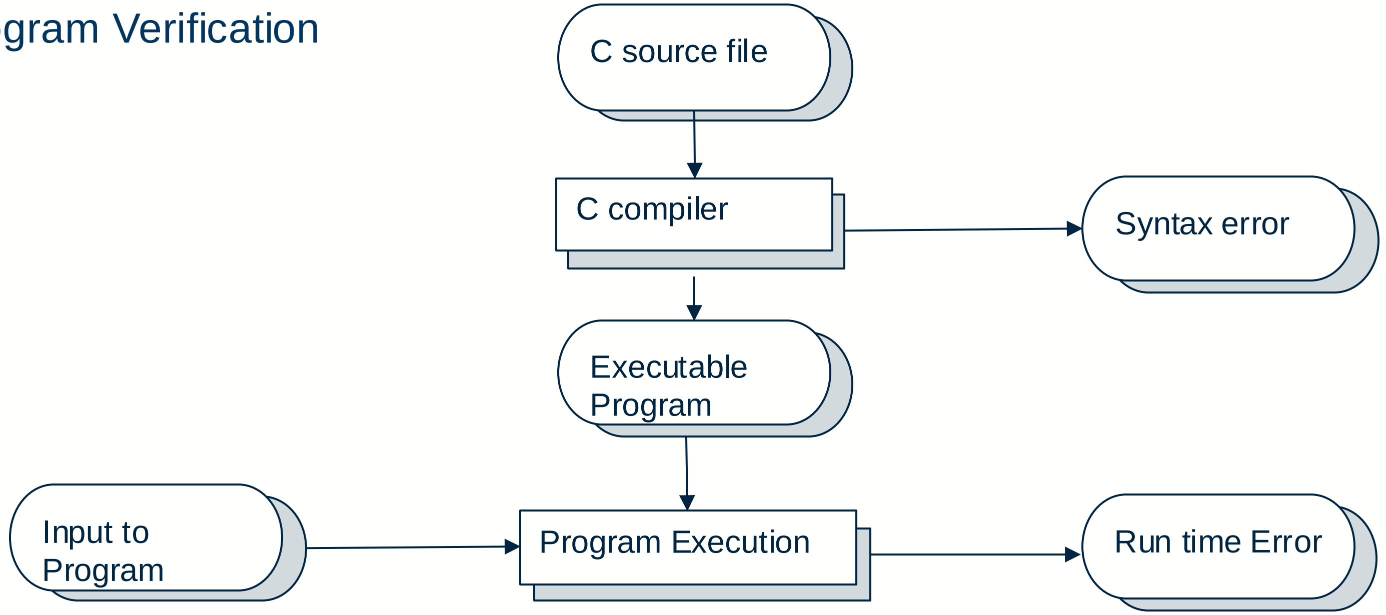
- Pattern Generalization/Abstraction
    - Use the same function for comparison with argument x, smallest or largest
        - count the data in the set and N variable x
- Algorithm Design

```
BEGIN
    $count \leftarrow 0$
    $x_{min} \leftarrow 0$
    $x_{max} \leftarrow 0$
    DO
        READ $x_{count+1}$
        IF $x_{count+1} < x_{min}$
            $x_{min} \leftarrow x_{count+1}$
        ENDIF
        $count \leftarrow count + 1$
    WHILE $count < N$
        $count \leftarrow 0$
    DO
        READ $x_{count+1}$
        IF $x_{count+1} > x_{max}$
            $x_{max} \leftarrow x_{count+1}$
        ENDIF
        $count \leftarrow count + 1$
    WHILE $count < N$
        PRINT $x_1 \cdots x_N, x_{max}, x_{min},$
END
```

Assume x has number smaller than 0

Problem Solving

- Program Verification



Problem Solving

- Syntax Error
  - Violation of the grammatical rule of language
    - Missing or misplaced semicolon
    - Illegal variable names e.g. use of invalid characters in a variable names
    - Misspelled keywords e.g. typing *floot* instead of *float*
    - Unmatched parentheses e.g. no. of left and right parentheses not matched in expression
    - Undefined variable names e.g. all variables have to be declared in C prior use
    - Missing subscripts in arrays
    - Declaring function arguments and return values of wrong type

Problem Solving

- Run-time Error
    - Error occur during the execution of a program
    - Caused by incorrect or incomplete algorithms that produce unexpected numbers or fail to allow for all possible input data
        - Indefinite result which occurs when a division by zero is attempted.
            - Overflow if value of y is zero

$$z = x/y$$

        - Complex result e.g. square root of a negative number
        - Overflow and underflow e.g. values greater than maximum value that can be stored in the variable of a given type.
        - Using invalid subscripts in arrays
        - Variables that have not been assigned a value , $a = b * c,\ a$ will be incorrect if b, c have not been assigned

University of Glasgow School of Computing Science

Problem Solving

- Logic errors
  - Result of faculty program design. Errors in logic can be insidious because the output from a program that contains logic errors may appear be to be deceptively correct
    - Incorrect condition in **if** statements
    - Incorrect conditions in while or do statements
    - Uninitialized or incorrect initialized variables
    - mistakes in the placement of semicolons\
    - failure to modify the loop controlling variable within a **while** or **do** loop
    - Unexpected end of data

University of Glasgow School of Computing Science

Problem Solving

- Debugging techniques
  - Use breakpoints feature if there is Integrated Development Environment (IDE)
  - Tracing using print(f) and scanf() statements
  - Hand simulation
  - Checking loops
  - Isolating sections of code
- Program Verification is a tedious process and is a job by its own. When testing, following should take note
  - Selection of testing data
    - Choose data that you know or can easily calculate the answer with minimum effort
    - Choose variety of different types of data which attempt to exercise many different paths of the programs

Problem Solving

- Each module should be completely tested before it is linked with the main module
- All boundary cases should be tested
- Null and illegal cases should be tested
- The modules should be combined and entire program especially intefaces between module should be tested
- Whenever a program is modified, all modules that have been changed should be tested including the whole program.

Problem Solving

- Documentation
  - User Documentation
    - All detail needed for end users to prepare the necessary input data, execute the program and interpret the results
    - Focus on input and output characteristics of the program & overview of the program
    - No details on the inner working of the program
  - Technical Documentation
    - For maintaining the program
    - All material needed for the programmer to change, correct and understand the program
    - Describe the
      - high level structure of the program (flowchart, pseudocode with description of each module interface, purpose)
      - Program listing (source code with comments)

Problem Solving

- Maintenance and Modification
    - Revision change
        - to correct bugs and errors
        - to meet changing user requirement
        - specification update due to expansion of scope
        - Hardware changes

Next Lecture

Lecture 3 : Data Type , Arithmetic Operations, Control Structures

#UofGWorldChangers

@UofGlasgow