SIT105 - Critical Thinking and Problem Solving for IT

Class 06

Part 1 - Steps in Algorithm Development

Part 2 - Building Algorithms

A/Prof. Xiao Liu

xiao.liu@deakin.edu.au

Review

UNIT WEEKLY ACTIVITIES

Week	Commencing	Topic
1	06 March	- Introduction - Problem Solving Techniques
2	13 March	- Claims and Issues
3	20 March	- Premises and Credibility
4	27 March	- Vagueness, Ambiguity, Generality and Fallacy
5	03 April	- Identifying and Analysing Arguments - Truth-Tables
6^	17 April	- Steps in Algorithm Development - Building Algorithms
7*	24 April	- Building and Checking Algorithms - Selection Statements
8	01 May	- Repetition Statements - Modularisation
9	08 May	- Module Cohesion - Module Coupling
10	15 May	- Flow Charts - Assertions

Claims and Issues

- A **claim** is a sentence where the main purpose is to communicate something that is either **true** or **false**.
- An **issue** is a **question** regarding whether a claim is true or false.
- Example claims and issues?

Credibility

- There are degrees of credibility for both:
  - Claims
  - Sources of claims

Claim	Source	Suspicion
High Credibility	High Credibility	Low
Lacks Credibility	High Credibility	Medium
High Credibility	Lacks Credibility	Medium
Lacks Credibility	Lacks Credibility	High

General, Vague & Ambiguous Statements

- Language is the foundation of our communication, so we need look closely to how it is being used
  - **A sentence, phrase or word is vague when it has no specific meaning or the meaning is unclear.**
  - **A sentence, phrase or word is ambiguous when it has more than one meaning or interpretation.**
  - A fallacy is a kind of error in reasoning or thinking.
  - ***Generality*** *is a source of vagueness and ambiguity.*

Quality and Test Plan for your Requirements

- Validate Requirements; inspect for common defects and able to answer **yes** to the following questions:
  - Is each requirement abstract?
  - Is each requirement unambiguous?
  - Is each requirement verifiable?
  - Is each requirement traceable to a user need?
  - Is each requirement realistic and technically feasible? may be hard to determine. Handy to compare benchmarks or system prototypes.
  - Collective properties
    - Are all the end-user needs addressed in the Requirements Specfn?
    - There should be no overlap or redundancy between requirements.
    - The Requirements Specification should not be self-contradictory.
- Characteristics of a Good Requirement
  - https://www.informit.com/articles/article.aspx?p=1152528&seqNum=4#

Long vs. Short Truth Table

- Long truth table
  - Complete map of all possible combinations
  - A truth table creater: https://www.cs.utexas.edu/~learnlogic/truthtables/
- Short truth table
  - You can think of the short truth table technique as like a game with permitted and forbidden moves. The objective of the game is to find a row out of all the rows in a full truth table which has all true premises and a false conclusion. Such a row, if it exists, would, of course, show the argument to be *invalid*. Thus the objective of the game is to prove the argument is **invalid.**

Part 1 Content

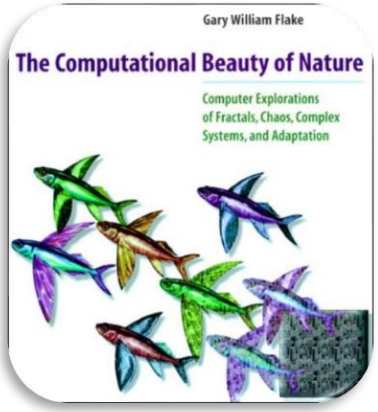
1. Programming/Algorithm Essentials
2. Program/Algorithm Development
3. References

QUOTE OF THE WEEK

To use: Apply shampoo to wet hair. Massage to lather, then rinse. Repeat.

A typical hair-washing algorithm that fails to stop! —

In our algorithms, programs and apps we must ***avoid*** *an* ***infinite loop!***



Quote by: Gary William Flake
Author of the book:
Computational Beauty of Nature

What is an Algorithm ?

What's an algorithm_ - David J. Malan.mkv

What are Essentials for Programming?

- **Data** – inputs/outputs
- **Algorithms** – sets of instructions
- **Modules, tasks, procedures, functions** – to perform some operation.
- **Order of tasks** – e.g. which function will be done first.
- **Associating data with tasks** – inputs into functions.
- **Refinement of data** – identifying relevant attributes and tasks/sub tasks

Refinement of Data

- Identifying relevant attributes
- What would a **student record** consists of?

  identification number

  name

  address

  course

  and others

What are the chunks of data involved?

Refinement: Attribute Data

- What would each of these parts consist of?

identification number

- **Year, unique number**

name

- **Given name, family name**

address

- **Street, suburb, postcode, …**

course

- **Code, title**

Can we split these into parts?

Refinement of Data

Have a think!

How could we break down the following examples of data refinement:

1. **A supermarket receipt consists of ...**
2. **A bank account consists of ...**
3. **The 'Critical Thinking' textbook consists of ...**

Refinement of Algorithms

Identifying sub tasks!

- For example, main task:
- **withdrawing cash from an ATM**

Can probably break this down into even smaller tasks

Sub tasks: The ATM will

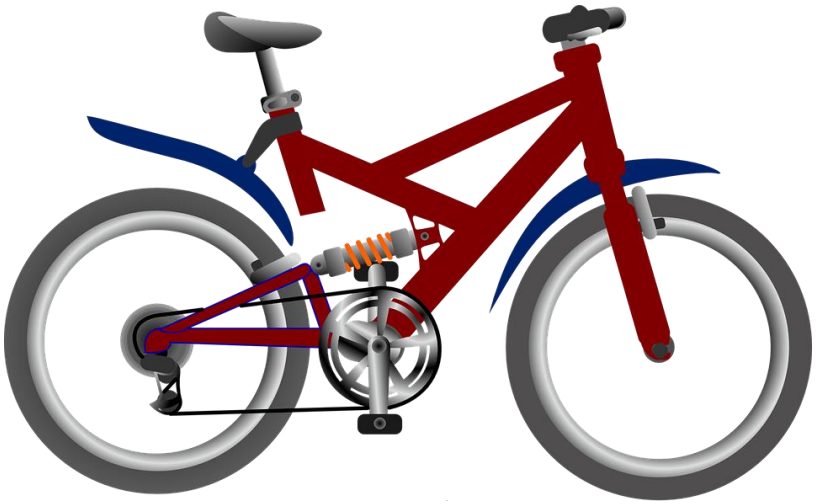
1. - machine waits for card
2. - prompt for PIN and obtain PIN
3. - check PIN using card information
4. - if PIN incorrect, terminate and reject card
5. - if PIN correct. proceed with the following
6. - prompt for and obtain cash amount
7. - prompt for and obtain account type
8. - process transaction
9. - eject card, cash and receipt

Refinement of Algorithms

Have a think!

What are the sub-tasks for these examples?:

1. **Riding a bicycle for at least 1 km consists of ...**
2. **Making a cup of coffee ...**
3. **Cooking rice …**



Steps in Program Development

1. Define the problem we are trying to solve
2. Outline the proposed solution
3. Develop an algorithm
4. Check that the algorithm is correct
5. Represent the algorithm in a programming language
6. Run the program
7. Documentation

Step 1. Define the Problem

Defining the problem is:

- to produce requirements (obtained from client).
  - What your program should do
- to produce specifications based on those requirements (obtained from developer).
  - How you plan to do it
- ***Difficult because*** :
  - we might overlook some characteristics of the problem
  - some requirements might be misinterpreted, vague, ambiguous, general, or conflict

Step 1. Define the Problem

Needs?

What does it do?

Produced?

The specification will normally include/describe:

1. **Input** data that the program requires
2. **Transformations/processing** to be performed on this data
3. **Output** data that the program must produce

and

1. Input data to each sub-task of the program
2. Transformations for each sub-task
3. Output data from each sub-task

Step 2. Outline the Solution

This is an **early draft** of the solution.

It consists of:

- tasks, subtasks
- data descriptions
- mainline logic (the main tasks in reasonable order)
- diagrams (UML, ER, DFD, hierarchy charts, ...)

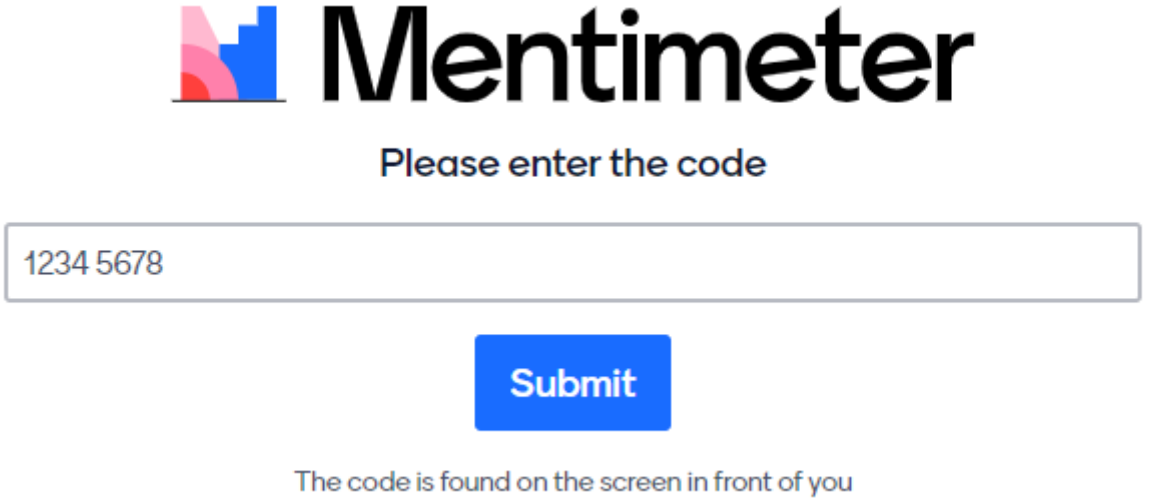
E.g. this task is to display a login screen

E.g. high level instructions or statements

Unified modelling language
Entity relationship diagrams
Data flow diagrams

Let’s go to www.menti.com

- What is the input, processing and output of a login screen?

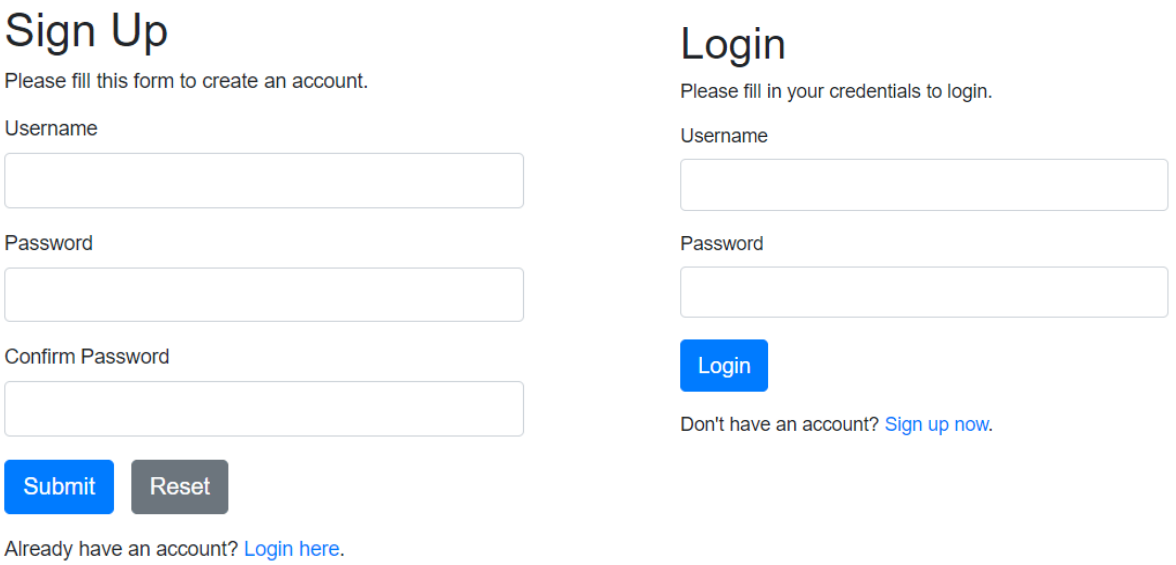


Login Screen

- Input
  - Input for username and password
    - Format checking?
- Processing
  - Checking the username and password
    - Connection to the database to retrieve the username and password
    - Check the status of the user account
    - Check the number of failed login attempts
    - …
- Output
  - Wrong username or password: login failed
  - Prompt the number of failed login attempts
  - Correct username and password: login successful -> redirect to the main screen

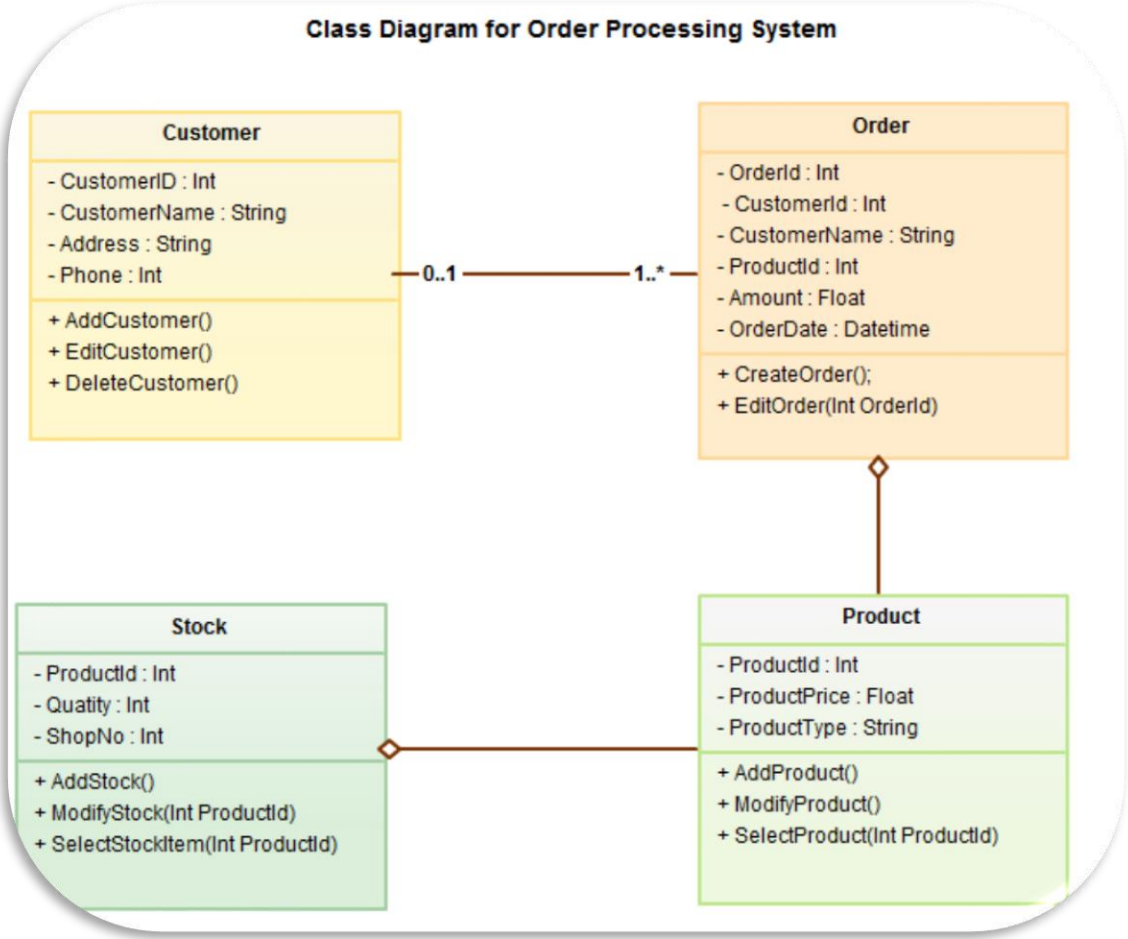
Let’s Look at a Real Example

- https://www.tutorialrepublic.com/php-tutorial/php-mysql-login-system.php



DIAGRAMS

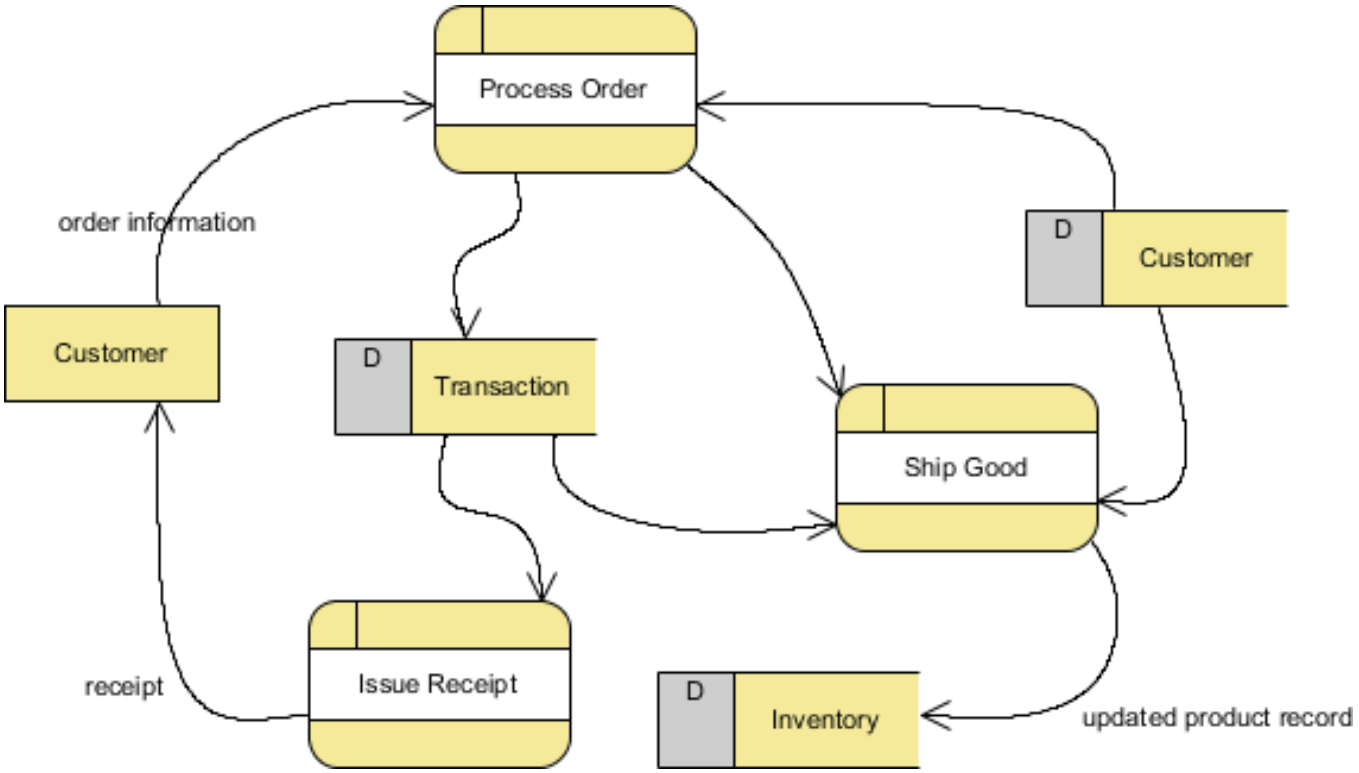
UML is a way of visualizing a software program using a collection of diagrams.



A customer may have 1 to many orders, an order may have at least 1 customer.

Ref: https://creately.com/blog/diagrams/class-diagram-relationships/

DIAGRAMS



A Data Flow Diagram (DFD) is a graphical representation of the "flow"
of data through an information system, modelling its process aspects.

Step 3. Develop an Algorithm

Produce an algorithm from Step 2 (Solution Outline) consisting of very specific instructions.

E.g., some specific instructions

```
PRINT "enter age"
READ age
IF (age > 12) and (age < 20) THEN
        PRINT "you are a teenager"
ELSE
        PRINT "you are not a teenager"
END-IF
```

Step 4. Check that the Algorithm is Correct

Usually called a **desk check**.

Use pencil and paper.

Go line-by-line and test the outputs to see if it matches what is expected

1. Choose a set of **test data**

Valid and invalid test data

2. Write down all **expected results**, independent of the algorithm
3. Using the algorithm, obtain **actual results** by evaluating each statement in detail, one statement at a time
   - *record all changes to variables*
   - *record all outputs*
4. **Compare** actual results from **3** to the expected results from **2**

Step 5. Convert the Algorithm to Code

- If the algorithm is correct, write the algorithm in the language of choice (C in the example below).
- Usually, each statement of the algorithm translates to a single statement of the chosen language.

```
printf("enter age");
scanf("%i", &age);
if(age > 12 && age < 20)
        printf("you are a teenager");
else
        printf("you are not a teenager");
```

Step 6. Run the Program

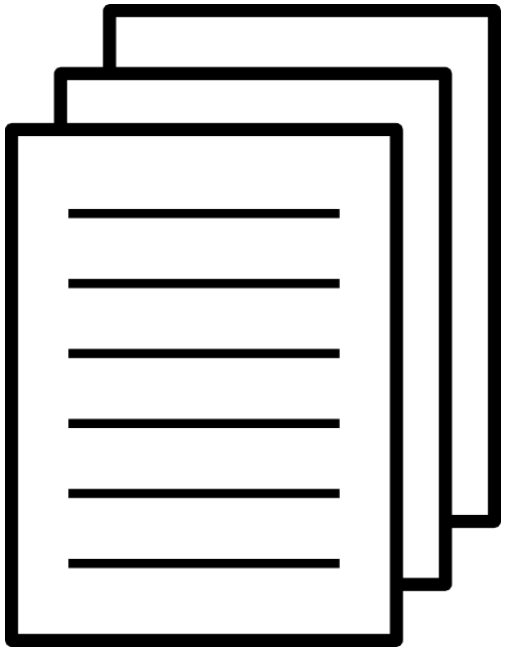
Another program to turn our code into a program

1. Compile the program.
2. Run the program using the test data.
3. Compare these actual results with expected results.
4. (If needed) Fix errors and go back to 1.
5. The program can be tested on real data.

Step 7. Documentation

We need to have a set of documentation to go along with our program, some of these are:

1. Program documentation
2. User documentation
3. Systems/Admin documentation



Program Documentation

- Intended to be read by a programmer
- Consists of:
  - algorithms
  - data dictionary (comprehensive description of variables in our program)
  - test data and results
  - diagrams (of task, subtasks, data) – UML/DFD/ERD
  - explanations placed within the program

```
// these are my comments
// coder: xxx
```

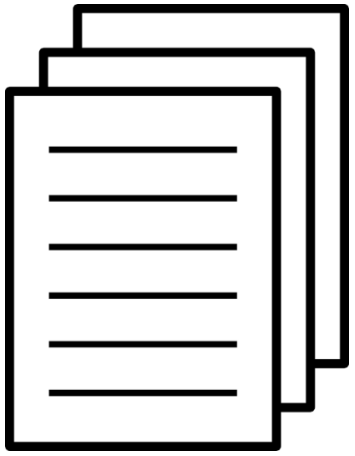
User Documentation

- Intended to be read by the user of the program
- User manual

- How the program operates. E.g.:
  - How to install
  - How to load
  - Keys used

Systems/Admin Documentation

- Intended to be read by a IT administrators and systems staff
- Installation instructions
- Systems instructions. E.g.
  - account creation, deletion, management
  - security management
  - updating versions
  - installing patches
  - backups



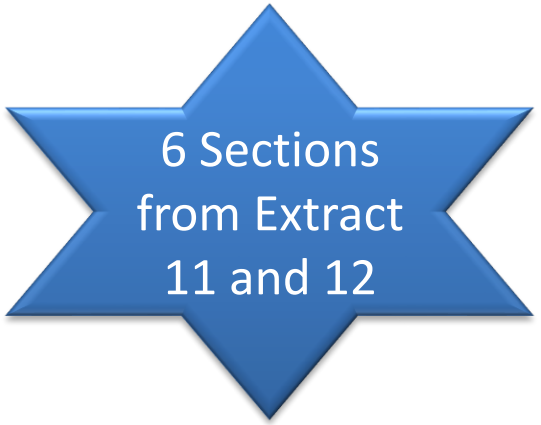
E.g. how to setup student accounts for Cloud Deakin ?

Part 2

Building Algorithms

Part 2 Content

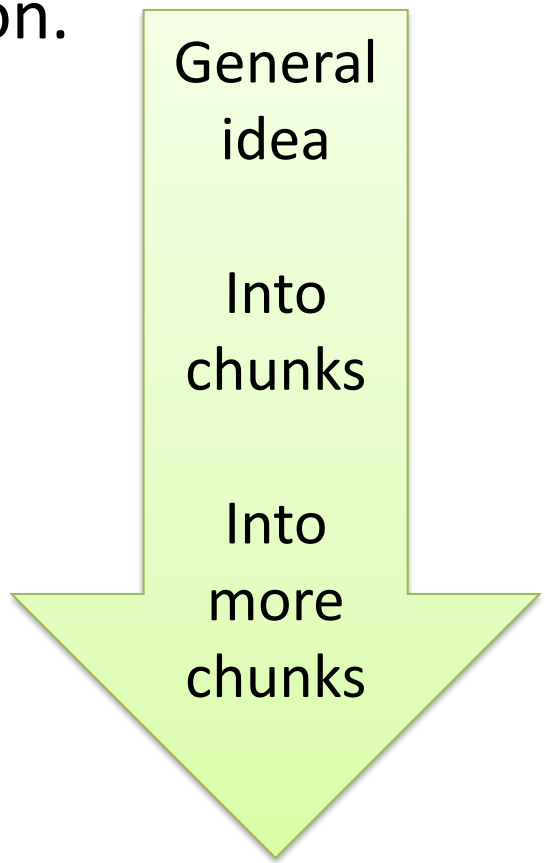
1. Top-down Development // section 1.3
2. Modular Development // section 1.3
3. Algorithms // section 1.4
4. Pseudocode // section 1.4
5. Writing Pseudocode // section 2.1
6. The Structure Theorem // section 2.3
7. References



Top-down Development

This is a method of steps to refine a solution.

- outline a general solution
- partition this general solution into more detailed sub-tasks
- continue this refinement process until all sub-tasks are **manageable**



That the particular chunk is small.
And easy to maintain

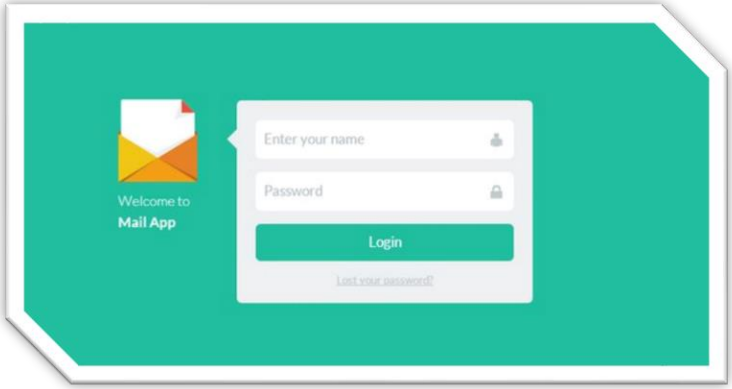
Top-down Development

For example, login task (e.g. login to gmail).

The login task may have the following sub-tasks.

1. Display login screen
2. Verify username and password (until correct)
3. Grant access

Continued.



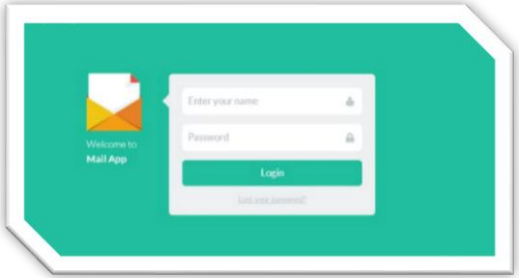
Some systems may only give you 3 chances!

Top-down Development

The **login sub-tasks** may have their own sub-tasks.

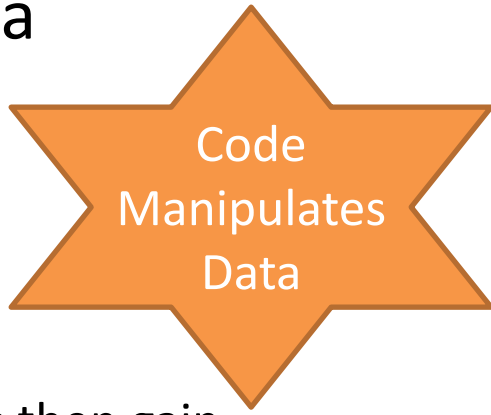
1. display login screen
   - display prompts
   - display text boxes
   - display buttons
2. verify username and password (until correct)
   - get username
   - get password
   - check username and password
3. grant access
   - remove login screen
   - display GUI for user

Broken down into sub tasks – 1,2,3



Modular Development

Modular development is the process of subdividing a computer program into separate sub-programs.

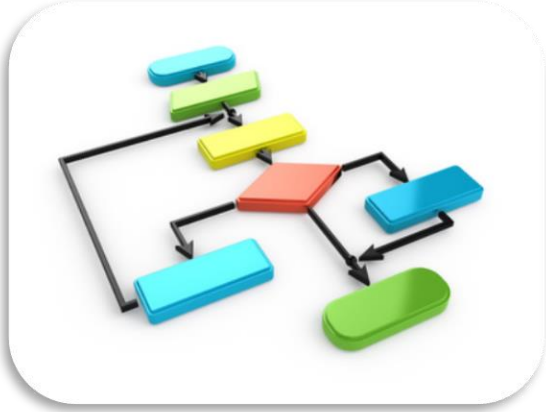


Modular development:

- A module performs a logical task
  - more so logical from a technological sense – if login success then gain entry
  - E.g. ***non logical*** – sending your password to every email to every account on the system after logging in.
- make modules small (easier to understand). ***1-20 lines good.***
- Place repeated code into a module – saves rewriting

Algorithms

- An *algorithm* is a set of instructions which specify how to perform a data manipulation task.



These instructions should:

- Be detailed!
- Be unambiguous!
- Terminate!
  (this is not a problem with everyday algorithms, but computer programs sometimes they get stuck in infinite loops)
  - E.g. with the quote of the week for washing hair

Pseudocode

Tool for writing algorithms, so that people can understand them!

Pseudocode is:

- a high level language
- a non-programming language
  - Can't build a compiler to build a program off this.
- used for writing (**abstract**) algorithms which:
  - *lack details of programming languages*
  - *are easy to understand*

Pseudocode

- Pseudocode consists of short, English phrases used to explain specific tasks within a program's algorithm.
- Pseudocode is a kind of structured English for describing algorithms.
- It allows the designer to focus on the logic of the algorithm without being distracted by details of language syntax.
- At the same time, the pseudocode needs to be complete.
- It describe the entire logic of the algorithm so that implementation becomes a mechanical task of translating line by line into source code.

The need for Pseudocode

The programming process is a complicated one.

- You must first understand the program specifications, then you need to organize your thoughts and create the program.
- This is a difficult task when the program is not easy.
- You must break the main tasks that must be accomplished into smaller ones in order to be able to eventually write fully developed code.
- Writing pseudocode will save you time later during the construction & testing phase of a program's development.

Pseudocode standards

- Each textbook and each individual designer may have their own personal style of pseudocode.
- Pseudocode is not a rigorous notation, since it is read by other people, not by the computer.
- There is no universal "standard" for the industry, but for instructional purposes it is helpful if we all follow a similar style.

Writing Pseudocode

There are six basic operations

1. Receive data - inputs
2. Output data – something that has been processed
3. Evaluate expressions – e.g. what is 1+1=
4. Assign a value to a variable – e.g. x=1 y=1
5. Select a group of statements
   - Group of statements – you want execute them sometimes based on some kind of event.
6. Repeat a group of statements
   - Repeat a bunch of statements more than once

1. Receiving Data

READ or GET obtains data from:

- The keyboard
- A file
- A socket

E.g.,

READ student_name

READ data FROM data_file

GET price1 price2 price3

2. Data Output

PRINT, WRITE or PUT causes data to be written to:

- the computer screen
- a file
- a socket

E.g.,

```
PRINT student_name
PRINT data TO data_file
WRITE price1 price2 price3
```

3. Evaluation of Expressions

- Arithmetic expressions (mathematical)

  price1 + price2 + price3

  x * x + 2 * x + 5

- Relational expressions (comparing things)

  x = 0

  12 < age < 20

- Logical expressions (and or and not)

  12 < age AND age < 20

  x > 0 AND y > 0

  p1 AND p2 AND (p3 OR p4) OR NOT (p5 AND p6)

Not limited to these!

4. Assigning a Value to a Variable

variable = expression

The expression on the **right side** of the assignment operator is evaluated and the result is stored in the *memory* location (the variable) named on the **left side** of the operator.

E.g.,

```
price1 = 100
price2 = 250
price3 = price2 * 1.20
total_price = price1 + price2 + price3
```

With a variable you can store data into memory.

5. Selection (IF statement)

Characteristics:

- keywords (usually IF, THEN, ELSE and ENDIF)
- condition
- groups of statements

Format 1:

```
IF condition THEN
    statement_1
    statement_2
    ...
ENDIF
```

If the condition evaluates to TRUE, execute these statements

5. Selection (IF statement)

Format 2:

```
IF condition THEN
    statement_1
    statement_2
      ...
ELSE
    statement_N
    statement_N+1
      ...
ENDIF
```

If the condition evaluates to TRUE, execute these statements

If the condition evaluates to FALSE, execute these statements

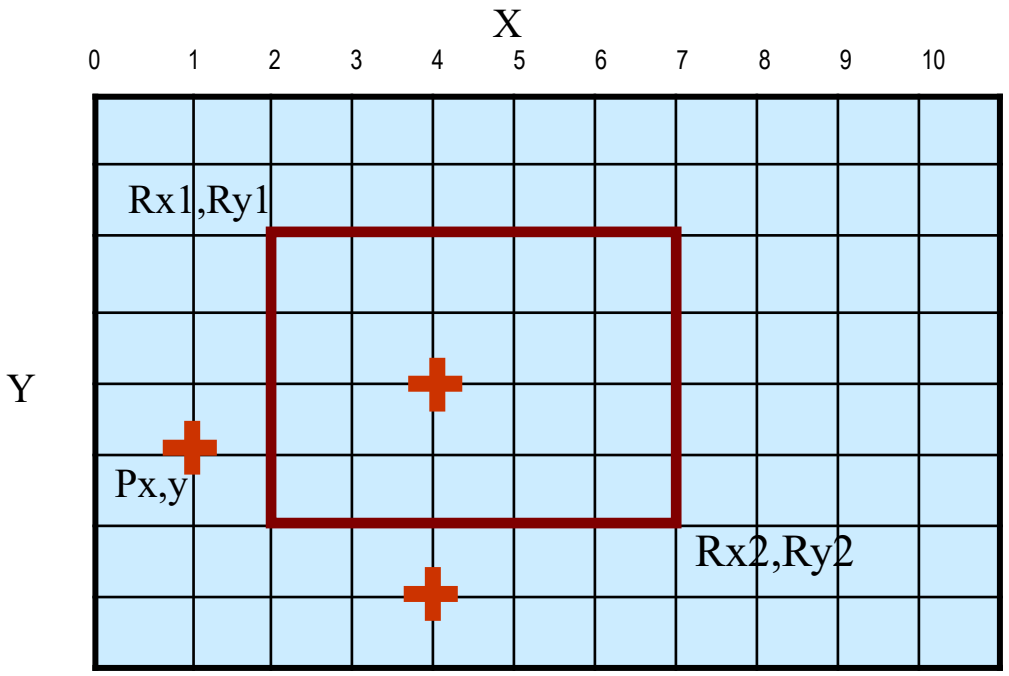
5. Selection (IF statement)

Examples.

```
IF traffic_light_color = green THEN
        drive through intersection
ENDIF
```

```
IF temperature > 25 THEN
        pack swimming gear
        go to beach
ELSE
        dress in jogging gear
        go for a run
ENDIF
```

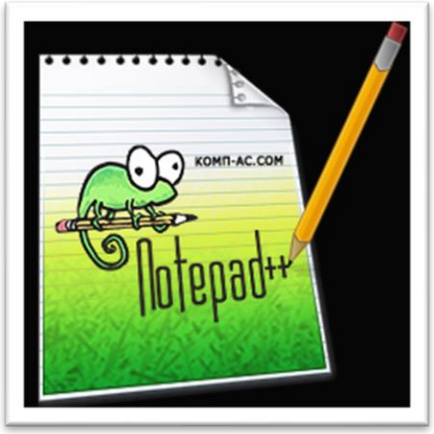
Problem



- Determine whether a given point is inside a given rectangle
- A Rectangle is defined by a point (Rx1,Ry1) and (Rx2,Ry2)
- A point is defined by P(x,y)

Pseudocode Editor

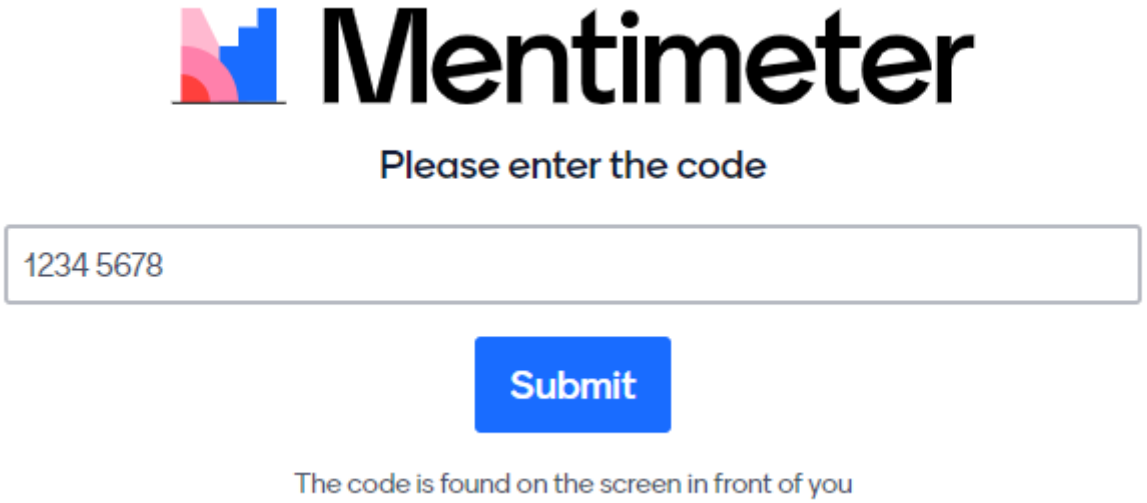
- Notepad ++



- UltraEdit

Let’s go to www.menti.com

- 5mins!



Demo on Pseudocode

6. Repetition (WHILE statement)

Characteristics:

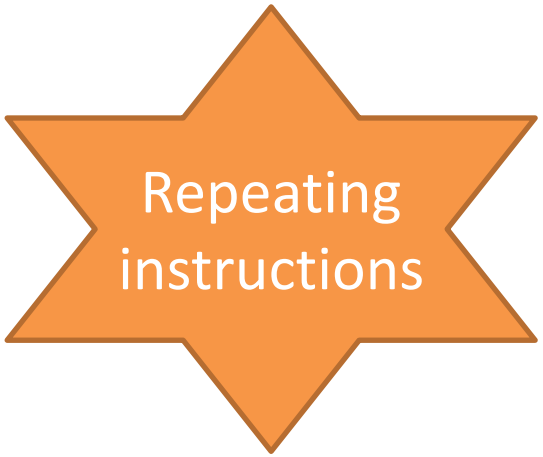
- keywords (WHILE, DO and ENDWHILE, or similar)
- condition
- one group of statements

Format:

```
WHILE condition DO
    statement_1
    statement_2
    ...
ENDWHILE
```

If the condition evaluates to TRUE, execute these statements

After executing these statements, repeat the whole WHILE statement



6. Example 1

Continually display 0, i.e., display 0000000000…

```
WHILE true DO
    WRITE 0          // display 0
ENDWHILE
```

When its false it will stop

6. Example 2

Continually display 0, i.e., display 0000000000…

```
x = 0
WHILE true DO
        WRITE x       // display the value of x
ENDWHILE
```

This time we assigned a value to a variable and used it.

6. Example 3

Display eleven zeros, i.e., 00000000000

```
x = 0
WHILE x <= 10 DO
    WRITE 0          // display 0
    x = x + 1        // add 1 to x
ENDWHILE
```

Keep repeating until X is less than or equal to 10.

Increment 1

6. Example 4

Display 0, 1, 2, 3, … 10

```
x = 0
WHILE x <= 10 DO
        WRITE x          // display x
        x = x + 1        // add 1 to x
ENDWHILE
```

Increment our variable X by 1 each time around. Then display it.

6. Example 5

Example: display 0, 1, 2, 3, … 10 and their square.

```
x = 0
WHILE x <= 10 DO
        y = x * x       // compute square
        WRITE x y       // display x and its square
        x = x + 1       // add 1 to x
ENDWHILE
```

Display 0 to 10 and then the result of 1^2, 2^2, 3^3

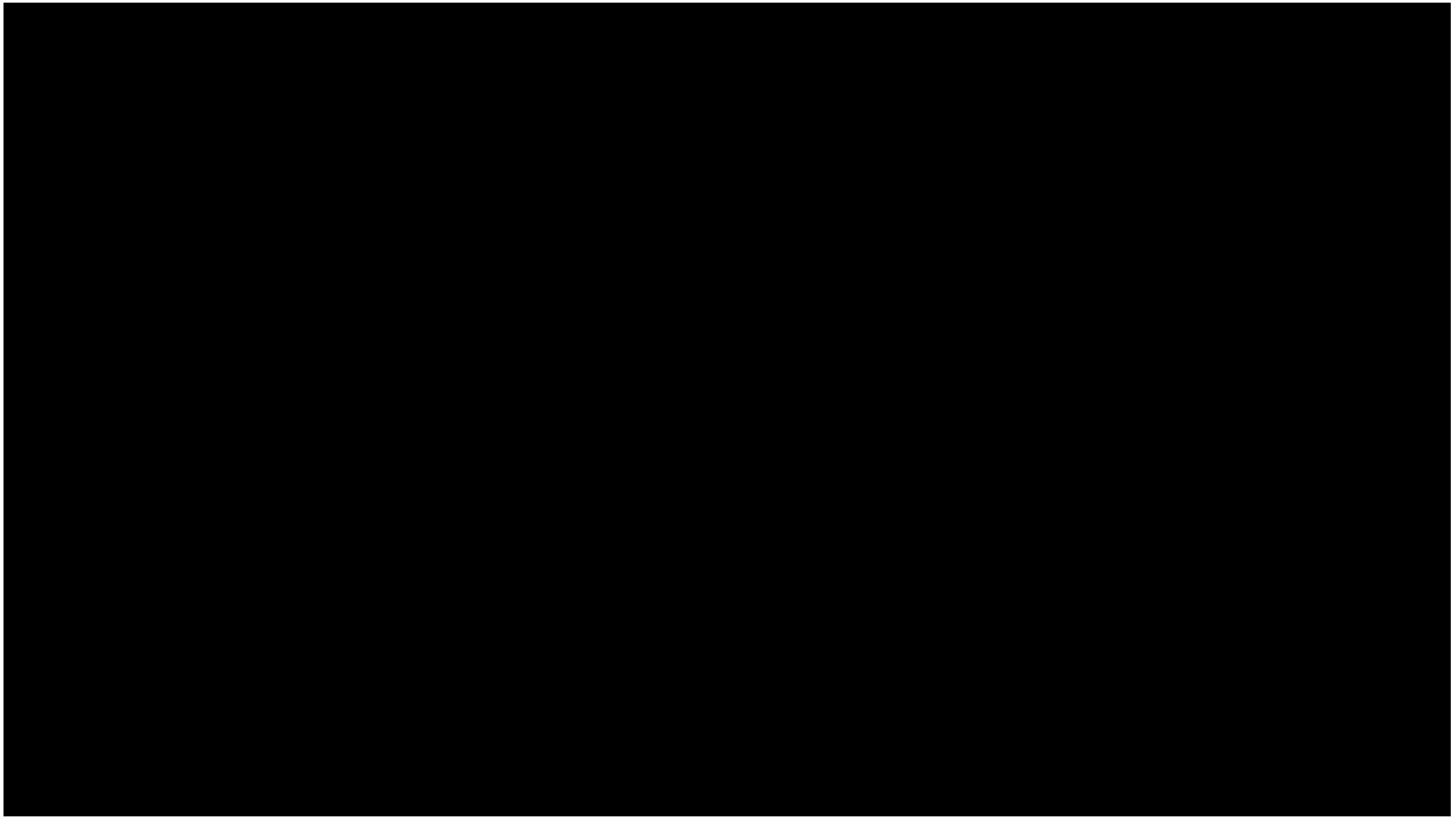
Structure Theorem

- ***The Structured Theorem forms the basic framework for structured programming.***

- There are only **three** code statements which are necessary to write **any** program/algorithm.

1. Sequence Statements (Formed by a sequence of one or more statements)
2. Selection Statements (IF statement)
3. Repetition Statements (WHILE statement)

Game time: can you solve the passcode riddle?



Source: https://ed.ted.com/lessons/can-you-solve-the-passcode-riddle-ganesh-pai

Can you solve the passcode riddle? 5mins

- In this dystopian world, your resistance group is humanity's last hope. Unfortunately, you've all been captured by the tyrannical rulers and brought to the ancient colosseum for their deadly entertainment. Before you're thrown into the dungeon, you see many numbered hallways leading outside. But each exit is blocked by an electric barrier with a combination keypad. You learn that one of you will be allowed to try to escape by passing a challenge while everyone else will be fed to the mutant salamanders the next morning. With her perfect logical reasoning, Zara is the obvious choice. You hand her a concealed audio transmitter so that the rest of you can listen along. As Zara is led away, you hear her footsteps echo through one of the hallways, then stop. A voice announces that she must enter a code consisting of three positive whole numbers in ascending order, so the second number is greater than or equal to the first, and the third is greater than or equal to the second. She may ask for up to three clues, but if she makes a wrong guess, or says anything else, she'll be thrown back into the dungeon. For the first clue, the voice says the product of the three numbers is 36. When Zara asks for the second clue, it tells her the sum of the numbers is the same as the number of the hallway she entered. There's a long silence. You're sure Zara remembers the hallway number, but there's no way for you to know it, and she can't say it outloud. If Zara could enter the passcode at this point, she would, but instead, she asks for the third clue, and the voice announces that the largest number appears only once in the combination. Moments later, the buzz of the electric barrier stops for a few seconds, and you realize that Zara has escaped. Unfortunately, her transmitter is no longer in range, so that's all the information you get. Can you find the solution?

Questions?