Computing COMP1

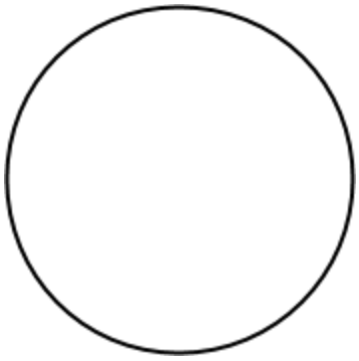
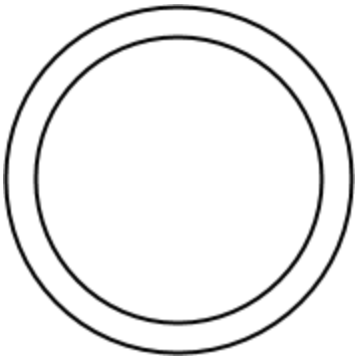
Fundamentals of Problem Solving

- Two main principles
    - **Abstraction**
        - How can complex ideas be communicated simply
        - How can we decompose problems logically
        - What is the right level of thinking for a problem
    - **Automation**
        - How can we automate the abstraction
- **Computation** the act or process of calculating or determining something by mathematical, logical or interactive methods
- **Computability** measures what can and cannot be computed
- **Computing** the study of natural and artificial information processes
- **An Algorithm** is a description, independant of any programming language, of a process that achieves some task. It is a step-by-step procedure for solving a problem
- **A Program** a description in a programming language of a process that achieves some useful result
- A **well define problem** has the following:
    - **Given** An initial situation
    - **Goal** A desired outcome
    - **Resources** The things that can be used to reach a goal
    - **Ownership** The people who takes responsibility for the problem
- The **stages of problem solving**
    - Understand what the problem is
    - Define what part of the problem we are going to solve and the boundaries
    - Plan the solution
    - Check solution is the optimum
- **Defining boundaries**
    - Establishing the limits or rules about what can and cannot be done when solving a problem. These limits are a type of constraint
- When making a plan of action you should ask these question
    - What strategies will you apply
    - What resources will you use
    - How will you use the resources
    - In what order will you use the resources
    - Are your resources adequate for the task
- **Top-down design**
    - Breaks a problem into smaller problems that are easier to work on
    - Each sub-problem can be thought of as a module or procedure
    - Diagram looks like a tree
- **Stepwise Refinement**
    - Process of breaking a problem down through successive steps into smaller problems

- Displayed as a structure table, an indented, numbered list of steps produced by stepwise refinement

Level 0	Define Problem
Level 1	Split into sub-problems 1 Subproblem 2 Subproblem
level 2	Further split 1.1 Subproblem 1.2 Subproblem 2.1 Subproblem 2.2 Subproblem ect ...

- **Finite State Machine** a machine that consists of a fixed set of possible states, with a set of allowable inputs that change the state and set of possible outputs
- **State** - What is happening at an exact instance in time
- **Events** - Something that happens in time, can trigger another state
- **Actions** - A task performed given a certain event
- **Transitions** - The change in state that occurs when an action takes place
- **State Transition Diagrams**

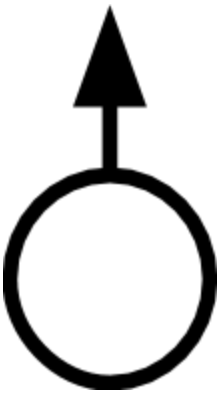
 State	 Accept (end) state
 Connector (where i is the input)	 Connector (where i is the input and k is the output)

- **State Transition Table** shows the effect on the current state of an FSM of particular inputs and any corresponding output
- **Decision Table** A table the shows the outcome for a given logical condition
- The **four main steps in algorithm design**
    - Identify the problem inputs
    - Identify the problem outputs
    - Determine the variables
    - Derive the algorithm that transforms inputs into outputs
- The **Three constructs** in algorithm design
    - **Sequence** consecutive steps or groups of steps processed one after another
    - **Selection** a decision-making step
    - **Iteration** a set of steps that are repeated until some condition is met
- **Assignment** is the operation that assigns a value to a variable
- **Structured English** a very restricted subset of the english language
- **Pseudocode** code that resembles a programming language but that uses less strict syntax to express an algorithm and is independent of any real programming language
- **Hand-trace, desk check or dry run** a careful step-by-step simulation on paper of how an algorithm would be executed by a computer

<u>Programming Structure</u>

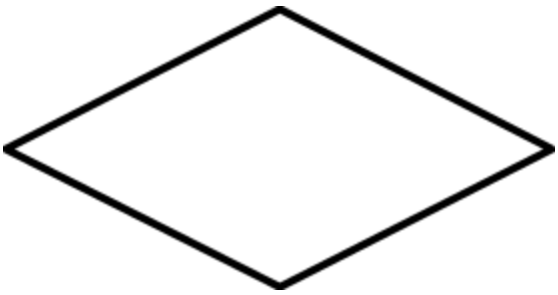
- **Structure Tables**
    - Breaking down a problem into subproblems and subsubproblems etc. and forming a numbered indented list of structured english
- **Hierarchy Charts**
    - Shows how top level modules call level 1 modules which call other modules
    - Looks like a tree
- **Structure Charts**
    - Is a hierarchy chart with interfaces and control information
    - How variable are passed between functions and repetitions
    - Symbols

Import Data



Export Data

Import and Export Data

 Major Decision	 Repeat

- **Features of Structured Programming**
    - Use meaningful identifiers
    - Use indentation
    - Procedures and functions with interfaces
    - A procedure should execute a single task
    - Use structure statements appropriately
- **Linear Search**
    - Step through area and test is true
- **Bubble Sort**
    - Same as in maths

<u>Checking for Errors</u>

- **Error types**
    - **Compilation Errors** - When a program does not compile due to syntax or lexical errors
    - **Run-time Error** - When the program crashes or get stuck in a loop
    - **Logic Error** - When the programmer has made a mistake and the program output is incorrect
- **Validation Check** a check made by a program to see that the data entered is reasonable
- **GIGO** if the user enter rubbish the answer they get will also be rubbish
- Types of Validation check
    - **Range Check** the enter value must be in a certain range
    - **Format Check** ensure the inputted data is in the correct format
    - **Length Check** checks if the string's length is expected
    - **Type Checks** entering text when a number is expects, read as string, check and then assign
    - **Lookup Check** query the input against a list of acceptable answers
    - **Presence Check** ensure the the user has filled out all essential sections
- **Check Digit** There is risk of error when entering in long strings of numbers, a digit is calculated and appended to the string
- Exception handling is used to trap runtime errors, try blocks

```
try
      dim ans as integer
```

```
      ans = num * 34
catch
      console.writeline("An ERROR OCCURRED, The inputted value was not
      an integer")
end try
```

- A system should be tested so that a robust system is created
- The purpose of testing is to reveal the presence of errors
- **Test strategies**
    - **Dry-run testing**
        - Manually step through code, often a trace table is used
    - **Black-box testing**
        - Testing without looking at the code, a function, part of code or the whole code is tested with specific test data, the output compared with the expected
    - **White-box testing**
        - Test each possible path through the program code
- **Types of test data**
    - **Normal data** - Data that is expected and should work
    - **Boundary data** - Data between two sets eg. valid and invalid or a fork
    - **Erroneous data** - Data that is wrong
- **Evidence of testing** something that shows the data entered and the result from the program eg. a screenshot

<u>Data Representation</u>

- A bit can be 1 or 0
- 8 bits in a byte, a group of bytes is a word
- Move values to left, double; to right half
- Two compliment have the same place values as unsigned but the leftmost bit is a negative place value

<u>Characters</u>

- **ASCII** uses 1 byte, **UNICODE** uses 2 bytes
- Parity bits are bits added to a bit pattern for error checking
- In odd parity the number of 1s is odd, in even the number is even
- If the parity changes after transmission there was an error in transmission
- **Hamming Code**
    - Can self correct single errors using few parity bits
    - All bit positions that are powers of two are used a parity bits
    - Even parity is used
    - General rule for position *n*: skip *n-1* bits, check *n* bits, skip *n* bits, check *n* bits …
    - By summing the incorrect parity bits the incorrect bit can be found
- **Grey Code**
    - Designed so that only one bit changes each time it is incremented
    - Saves energy

    - Good for large chips with different clock speed as otherwise may get misinterpreted

    - Calculated by
        - First bit is the same
        - Second grey bit is the first binary bit XOR the second binary bit

Images

- A **pixel** is the smallest addressable area in an image
- A **bitmap image** is created when pixels of an image are mapped to positions in memory that store binary codes representing colours
    - When scaled will pixelate
    - Photographs with lots of continuous data will take up less memory are a bitmap
- **Resolution** is the number of pixels in given area (ppi)
- **Screen Resolution** is the number of pixels in horizontal direction by the number of pixels in the vertical direction
- **Colour depth** is the number of bit used to represent the color of a single pixel
- **Vector graphics** records geometric and other information about the objects that make up the image
    - Geometric images will take up less memory
    - Geometric images will load faster than bitmap images
    - When scaled will not distort
- **Objects** are components of vector graphics, eg. a line, a rectangle or a circle
- A drawing list is the list of drawing commands that create the vector graphic
- **Property** of an object is a description of an aspect of it eg. size, typeface, colour, radius, position
- **Data Compression** puts the data into a smaller number of bytes than the data would otherwise occupy
- **Run-length Encoding** if three or more consecutive memory locations have the same colour, it is compressed into two bytes, the first the number of consecutive locations and the second the value
    - This is lossless
- **Lossy Compression** compresses the image by removing data
    - Reduces the resolution of the background
    - Uses few bits of colour to which the human eye is less sensitive
    - Fewer bits of low-intensity colours

Sound

- **Analogue data** data that varies in a continuous manner
- **Analogue signal** an electrical signal that varies in a continuous manner
- **Digital data** data that takes the form of discrete values
- **Digital signal** an electrical signal with voltage changes that are abrupt or in discrete steps
- **Analogue to digital converter (ADC)** samples the sound many times per second and saves the samples are binary numbers

- **Pulse Amplitude Modulation (PAM)** The analogue signal is sampled and recorded as a decimal number
- **Pulse Code Modulation (PCM)** The analogue wave is sampled and stored as a binary number
- **Sampling rate** is the number of times a wave is sampled in a second
- **Sampling resolution** is the number of bits the sampled data is stored in
- **Analogue to digital conversion process**
    1. Analogue signal is sampled at regular intervals, this is pulse amplitude modulation (PAM)
    2. The samples are then quantised , the height of each PAM sample is approximated, this produces the PMC data
    3. The PMC data is then encoded into binary
- **Nyquist's theorem** sound should be sampled at a frequency at least twice that of the highest frequency in the sampled signal
- The higher sampling resolution will cause the sound to be truer however it will also take up more memory
- **File Formats**
    - WAV
    - MPEG, compressed
- **Musical Information Digital Interface (MIDI)** a way of representing the sounds made by an instrument
- **Synthesise Sound** use digital means to generate audio signals resembling instrument sounds or the human voice
- **Streaming audio** data is downloaded to a buffer continuously as the player reads the data from the buffer
- Sounds can be edited using software on the computer

<u>Stages of Development</u>

- **System** is a whole composed of parts in an orderly arrangement
- **Manual System** is one that does not involve computers
- **The stages of system design are:**
    - **Analysis,** methods for analysis
        - Interviews
        - Observation
        - Questionnaires
        - Examination of documentation
    - **Design Phase**, must be specified
        - Required hardware and software
        - Required inputs and outputs, input methods, printed reports etc.
        - UIs, GUIs
        - Data files, size
        - Algorithms to be used
        - Test plan
    - **Implementation**

        - Developing hardware and software
        - Installation of hardware and software
        - Preparing files
        - Training users
        - Writing documentation
    - **Testing Phase**
        - Alpha and Beta testing
        - Dry run
        - White and black box
    - **Evaluation**
        - Does it work correctly
        - Can it be improved
- **System Maintenance** updating a program to correct faults and improve features