



COMP110: Principles of Computing

## **2: Algorithms**

# **Programming languages and paradigms**

# What is a programming language?

- ▶ A **program** is a sequence of instructions for a computer to perform a specific task
- ▶ A **programming language** is a formal language for communicating these sequences of instructions

# Which is the best programming language?

- ▶ There is no “best” programming language
- ▶ There are hundreds of programming languages, each better suited to some tasks than others
- ▶ Sometimes your choice is dictated by your choice of platform, framework, game engine etc.
- ▶ To become a better programmer (and maximise your employability) you should learn several languages (but one at a time!)

# Low vs high level

- ▶ **Low level languages** give the programmer direct control over the hardware
- ▶ **High level languages** give the programmer **abstraction**, hiding the details of the hardware
- ▶ High level languages trade efficiency for ease of programming
- ▶ Lower level languages were once the choice of game programmers, but advances in hardware mean that higher level languages are often a better choice

# Programming paradigms

- ▶ **Imperative**: program is a simple sequence of instructions, with **goto** instructions for program flow
- ▶ **Structured**: like imperative, but with **control structures** (loops, conditionals etc.)
- ▶ **Procedural**: structured program is broken down into **procedures**
- ▶ **Object-oriented**: related procedures and data are grouped into **objects**
- ▶ **Functional**: procedures are treated as mathematical objects that can be passed around and manipulated
- ▶ **Declarative**: does not define the control flow of a program, but rather defines logical relations

# Which paradigm?

- ▶ **Imperative** and **structured** languages are mainly of historical interest
- ▶ Most commonly used languages today are a mixture of **procedural** and **object-oriented** paradigms, with many also incorporating ideas from **functional** programming
- ▶ Purely **functional** languages are mainly used in academia, but favoured by some programmers
- ▶ Purely **declarative** languages have uses in academia and some special-purpose languages

# Machine code

```
00000000 40 5a 90 00 03 00 00 00 00 00 00 00 00 00 00 00
00000010 b8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 30 01 00 00
00000040 0e 1f ba 0e 00 04 09 cd 21 b0 01 4c cd 21 54 60
00000050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f
00000060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20
00000070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00
00000080 75 99 69 bc 31 78 07 ef 31 78 07 ef 31 78 07 ef
00000090 a2 b6 9f ef 3c 78 07 ef 2a 65 99 ef 70 78 07 ef
000000a0 2a 65 ac ef 7f 78 07 ef 2a 65 ad ef ec 78 07 ef
000000b0 5e 8e ac ef 32 78 07 ef 16 3e 6a ef 35 79 07 ef
000000c0 f2 7f 58 ef 33 78 07 ef f2 7f 5a ef 35 78 07 ef
000000d0 f2 7f 67 ef 30 78 07 ef 38 00 83 ef 30 78 07 ef
000000e0 38 00 94 ef 14 78 07 ef 31 78 06 ef 55 7a 07 ef
000000f0 2a 65 a8 ef 02 79 07 ef 2a 65 9c ef 30 78 07 ef
00001000 2a 65 9d ef 30 78 07 ef 2a 65 9a ef 30 78 07 ef
00001100 52 69 63 68 31 78 07 ef 00 00 00 00 00 00 00 00
00001200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00001300 50 45 00 00 4c 01 03 00 5f 60 9a 57 00 00 00 00
00001400 00 00 00 00 e0 00 03 01 0b 01 0a 00 00 70 10 00
00001500 00 40 00 00 00 30 37 00 a0 25 40 00 00 40 37 00
00001600 00 30 40 00 00 00 40 00 00 10 00 00 00 02 00 00
00001700 05 00 01 00 00 00 00 00 05 00 01 00 00 00 00 00
00001800 00 70 40 00 00 10 00 00 00 00 00 00 02 00 00 01
00001900 00 00 00 10 00 00 10 00 00 00 10 00 00 10 00 00
00001a00 00 00 00 00 10 00 00 00 50 3d 26 00 4b 00 00 00
00001b00 70 62 40 00 00 04 00 00 00 30 40 00 70 32 00 00
00001c00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- ▶ Programs are represented as sequences of **numbers** specifying **machine instructions**
- ▶ More on this later in the module
- ▶ Nobody has actually written programs in machine code since the 1960s...



# Assembly language

```
section      .text
global      _start

_start:

    mov     edx,len
    mov     ecx,msg
    mov     ebx,1
    mov     eax,4
    int     0x80

    mov     eax,1
    int     0x80

section      .data

msg         db  'Hello, world!',0xa
len         equ  $ - msg
```

- ▶ Each line of assembly code translates **directly** to an instruction of machine code
- ▶ Commonly used for games in the 70s/80s/90s, but hardly ever used now
- ▶ Allows very fine control over the hardware...
- ▶ ... but difficult to use as there is no **abstraction**
- ▶ Also not portable between CPU architectures

# C++

- ▶ Initially an object-oriented extension for the procedural language C
- ▶ Low level (though higher level than assembly)
- ▶ Used by developers of game engines, and games using many popular “AAA” engines (Unreal, Source, CryEngine, ...)
- ▶ Also used by developers of operating systems and embedded systems, but falling out of favour with other software developers

```
#include "stdafx.h"
#include "GameObject.h"
#include "CoinGame.h"

GameObject::GameObject(CoinGame* game, Texture* sprite)
: game(game), sprite(sprite), isDead(false)
{
    x = rand() % CoinGame::WINDOW_WIDTH;
    y = rand() % CoinGame::WINDOW_HEIGHT;
}

GameObject::~GameObject()
{
}

void GameObject::render(SDL_Renderer* renderer)
{
    sprite->render(renderer, x, y, CoinGame::SPRITE_SIZE, CoinGame::SPRITE_SIZE);
}

bool GameObject::checkCollision(int otherX, int otherY)
{
    double distance = sqrt(pow(otherX - x, 2) + pow(otherY - y, 2));
    return distance < CoinGame::SPRITE_SIZE / 2;
}
```

# High level languages

Often favoured by smaller indie teams for rapid development

- ▶ C# (XNA, Unity)
- ▶ Python (EVE Online, Pygame, Ren'py)
- ▶ JavaScript (HTML5 browser games)
- ▶ ActionScript (Flash games)
- ▶ Objective-C, Swift (iOS games)
- ▶ Java (Minecraft, Android games)

There are many others, but these are the most commonly used in game development

# Scripting languages

Many games use scripting languages in addition to their main development language

- ▶ Lua (many AAA games)
- ▶ Bespoke languages (many AAA games)

Some game engines have their own scripting language

- ▶ UnrealScript, Blueprint (Unreal Engine)
- ▶ GML (GameMaker)

# Visual programming languages



Based on connecting graphical blocks rather than writing code as text

- ▶ Scratch (used for teaching in school)
- ▶ Lego Mindstorms
- ▶ Blueprint (Unreal)



Note: despite the name,  
Microsoft Visual Studio is **not** a  
visual programming environment!

# Special purpose languages

- ▶ SQL (database queries)
- ▶ GLSL, HLSL (GPU shader programs)
- ▶ LEX, YACC (script interpreters)

# Markup languages

Not to be confused with programming languages...

- ▶ HTML, CSS (web pages)
- ▶ LaTeX, Markdown (documentation)
- ▶ XML, JSON (data storage)

# Which programming language is most popular?

`http://github.info`



# “Family tree” of programming languages

<https://www.levenez.com/lang/lang.pdf>

# Algorithms

# What is an algorithm?

A **sequence of instructions** which can be followed **step by step** to perform a **(computational) task**.

# Algorithms historically

- ▶ Named after Muhammad ibn Musa al-Khwarizmi (c. 780–850), Persian mathematician
- ▶ Used in mathematics to describe steps for calculations
  - ▶ E.g. Euclid's algorithm for finding the greatest common divisor of two numbers
- ▶ Computers developed as machines for carrying out mathematical algorithms

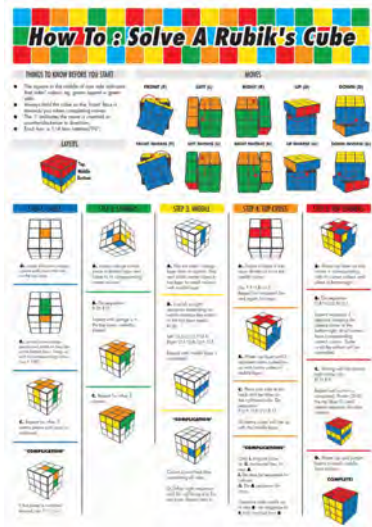
# Programs vs algorithms

- ▶ A program is **specific** to a particular programming language and/or machine
- ▶ An algorithm is **general**
- ▶ An algorithm must be **implemented** as a program before a computer can run it
- ▶ An algorithm generally performs **one task**, whereas a program may perform **many**
  - ▶ E.g. Microsoft Word is not an algorithm, but it implements many algorithms
  - ▶ E.g. it implements an algorithm for determining where to break a line of text, how much space to add to centre a line, etc.

# Algorithms outside computing

- 1 Preheat the oven to 180C, gas 4.
- 2 Beat together the eggs, flour, caster sugar, butter and baking powder until smooth in a large mixing bowl.
- 3 Put the cocoa in separate mixing bowl, and add the water a little at a time to make a stiff paste. Add to the cake mixture.
- 4 Turn into the prepared tins, level the top and bake in the preheated oven for about 20-25 mins, or until shrinking away from the sides of the tin and springy to the touch.
- 5 Leave to cool in the tin, then turn on to a wire rack to become completely cold before icing.
- 6 To make the icing: measure the cream and chocolate into a bowl and carefully melt over a pan of hot water over a low heat, or gently in the microwave for 1 min (600w microwave). Stir until melted, then set aside to cool a little and to thicken up.
- 7 To ice the cake: spread the apricot jam on the top of each cake. Spread half of the ganache icing on the top of the jam on one of the cakes, then lay the other cake on top, sandwiching them together.
- 8 Use the remaining ganache icing to ice the top of the cake in a swirl pattern. Dust with icing sugar to serve.

# Algorithms outside computing



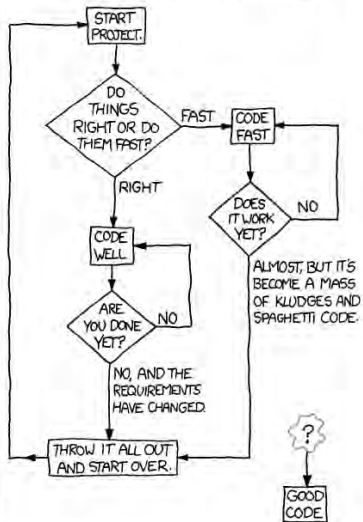
# Why algorithms?

- ▶ Allow for common computations to have **common solutions**
- ▶ **Algorithm strategies** give widely applicable approaches for solving problems
- ▶ Can **prove** mathematically that an algorithm does what it is supposed to
- ▶ Can reason about the **complexity** (time, space etc) of an algorithm — and place **lower bounds** on the best possible algorithm
- ▶ **Computability** theory lets us reason about what computations are and are not possible

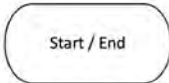


# Flowcharts

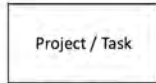
## HOW TO WRITE GOOD CODE:



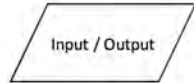
# Flowchart symbols



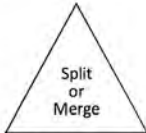
The start or end of a workflow.



Process or action.



Data: Inputs to, and outputs from, a process.



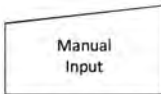
Upright indicates a process split, inverted indicates a merge of processes.



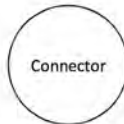
Decision point in a process or workflow.



Document or report.



Prompt for information, manually entered into a system.

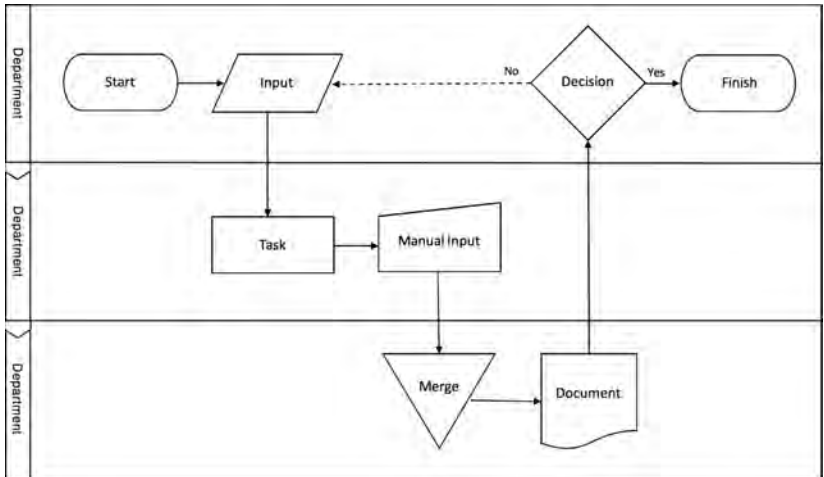


Used to connect one part of a flowchart to another.



Connector used to connect one page of a flowchart to another.

# Swimlanes



# Software for drawing flowcharts

Intended for drawing flowcharts:

- ▶ Gliffy <https://www.gliffy.com>
- ▶ LucidChart
- ▶ Microsoft Visio

Can draw flowcharts:

- ▶ Microsoft PowerPoint
- ▶ Google Docs

If you're desperate:

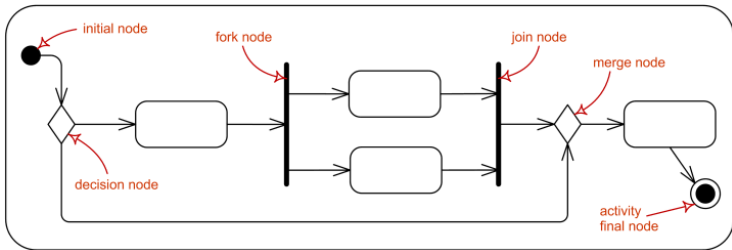
- ▶ Any drawing package (Inkscape, Adobe Illustrator, Apple Keynote, ...)
- ▶ MS Paint
- ▶ Pen and paper

# Activity

- ▶ In **groups of 2-3**
- ▶ **Draw** a flowchart for **logging into Facebook**
- ▶ Include at least two swimlanes: **the user's browser/device** and **the Facebook server**
- ▶ Post your flowchart to **#comp110** on **<https://falmouthgamesacademy.slack.com>**

# UML activity diagrams

- ▶ Modern counterpart of flowcharts
- ▶ UML = Unified Modeling Language — defines 14 types of diagram to represent various aspects of computing systems, of which activity diagrams are one



**Pseudocode**



# Pseudocode

Flowcharts are useful, but...

- ▶ Can be time-consuming to draw
- ▶ Do not reflect structured programming concepts well

**Pseudocode** expresses an algorithm in a way that looks more like a structured program

# Pseudocode example

```
print "How old are you?"  
read age  
if age < 13 then  
    print "You are a child"  
else if age < 18 then  
    print "You are a teenager"  
else  
    print "You are an adult"  
end if
```

# Pseudocode example

```
sum ← 0                                ▷ initialisation
for i in 1,...,9 do
    sum ← sum + i
end for
print sum                             ▷ print the result
```

<https://socrative.com>, room code FALCOMPED:  
what would this print?

# Pseudocode example

```
 $a \leftarrow 1$                                 ▷ initialisation  
while  $a < 100$  do  
     $a \leftarrow a \times 2$   
end while  
print  $a$                                 ▷ print the result
```

<https://socrative.com>, room code FALCOMPED:  
what would this print?

# Formatting pseudocode

- ▶ Pseudocode is a **communication tool**, not a **programming language**
- ▶ Important: **clear, concise, unambiguous, consistent**
- ▶ **Not** important: adhering to a strict set of style guidelines, ensuring direct translatability to your chosen programming language

# Level of abstraction

Whether working with flowcharts or pseudocode, choose your **level of abstraction** carefully

# Level of abstraction: Good

Fill kettle

Turn kettle on

Put instant coffee in mug

**if** sugar wanted **then**

    Add sugar

**end if**

Wait for kettle to boil

**if** milk wanted **then**

    Pour water to  $\frac{4}{5}$  full

    Add milk

**else**

    Fill mug with water

**end if**

Stir

# Level of abstraction: Not so good

Position kettle beneath tap

Turn tap on

**while** water is below halfway point **do**

    Wait

**end while**

Turn tap off

Place kettle on base

Press power button

...



# Level of abstraction: Silly

Place right palm on kettle handle

Bend fingers on right hand

Lift arm upwards

**while** tap spout is not directly above kettle **do**

    Move arm to the right

**end while**

Place left palm on tap handle

Bend fingers on left hand

Rotate left hand

...

Level of abstraction: also silly

Make a cup of coffee

# Activity

A number guessing game: The computer chooses a number between 1 and 20 at random. The player guesses a number. The computer says whether the guessed number is “too high”, “too low” or “correct”. The game ends when the correct number is guessed, or after 5 incorrect guesses.

- ▶ In **groups of 2-3**
- ▶ **Write** pseudocode for the number guessing game
- ▶ **Post** your pseudocode on Slack
- ▶ Tip: type ` ` (top left key on your keyboard) **before and after** your pseudocode to preserve indentation and line breaks!