Object Orientation

Programming Fundamentals

Dr Kevin Chalmers

School of Computing
Edinburgh Napier University
Edinburgh

k.chalmers@napier.ac.uk

Outline

Outline

Outline

Outline

Outline

Historical Perspectives

- New approaches to programming develop over time

Historical Perspectives

- New approaches to programming develop over time
- There are a number of reasons for this. Some are:

Historical Perspectives

- New approaches to programming develop over time
- There are a number of reasons for this. Some are:
  1. User Interface (UI) / User Experience (UX)

Historical Perspectives

- New approaches to programming develop over time
- There are a number of reasons for this. Some are:
  1. User Interface (UI) / User Experience (UX)
     - A little bit like asking an artist to come up with a new hammer design

Historical Perspectives

- New approaches to programming develop over time
- There are a number of reasons for this. Some are:
  1. User Interface (UI) / User Experience (UX)
     - A little bit like asking an artist to come up with a new hammer design
  2. Particular approach to solving a problem

Historical Perspectives

- New approaches to programming develop over time
- There are a number of reasons for this. Some are:
  1. User Interface (UI) / User Experience (UX)
     - A little bit like asking an artist to come up with a new hammer design
  2. Particular approach to solving a problem
     - Write less code for that problem. *Domain Specific Language*

Historical Perspectives

- New approaches to programming develop over time
- There are a number of reasons for this. Some are:
  1. User Interface (UI) / User Experience (UX)
     - A little bit like asking an artist to come up with a new hammer design
  2. Particular approach to solving a problem
     - Write less code for that problem. *Domain Specific Language*
  3. Syntactic sugar

Historical Perspectives

- New approaches to programming develop over time
- There are a number of reasons for this. Some are:
  1. User Interface (UI) / User Experience (UX)
     - A little bit like asking an artist to come up with a new hammer design
  2. Particular approach to solving a problem
     - Write less code for that problem. *Domain Specific Language*
  3. Syntactic sugar
  4. Itch scratching

Historical Perspectives

- New approaches to programming develop over time
- There are a number of reasons for this. Some are:
  1. User Interface (UI) / User Experience (UX)
     - A little bit like asking an artist to come up with a new hammer design
  2. Particular approach to solving a problem
     - Write less code for that problem. *Domain Specific Language*
  3. Syntactic sugar
  4. Itch scratching
- So we have Machine Code $\rightarrow$ Assembly $\rightarrow$ Procedural $\rightarrow$ $\{$Object-Oriented$|$Functional$|$Logical$|\ldots\}$

Historical Perspectives

- New approaches to programming develop over time
- There are a number of reasons for this. Some are:
  1. User Interface (UI) / User Experience (UX)
     - A little bit like asking an artist to come up with a new hammer design
  2. Particular approach to solving a problem
     - Write less code for that problem. *Domain Specific Language*
  3. Syntactic sugar
  4. Itch scratching
- So we have Machine Code $\rightarrow$ Assembly $\rightarrow$ Procedural $\rightarrow$ $\{$Object-Oriented$|$Functional$|$Logical$|\ldots\}$
- **Why???**

Pictorial History of Programming UI

- Trend towards higher-level languages and better interfaces/ways to solve problems:

Pictorial History of Programming UI

- Trend towards higher-level languages and better interfaces/ways to solve problems:
  - Early machine e.g. Colossus, Manchester Baby

Pictorial History of Programming UI

- Trend towards higher-level languages and better interfaces/ways to solve problems:
  - Early machine e.g. Colossus, Manchester Baby
  - Terminal-Server and Batch Processing

Pictorial History of Programming UI

- Trend towards higher-level languages and better interfaces/ways to solve problems:
  - Early machine e.g. Colossus, Manchester Baby
  - Terminal-Server and Batch Processing
  - Keyboard, Screen with local machine

Pictorial History of Programming UI

- Trend towards higher-level languages and better interfaces/ways to solve problems:
  - Early machine e.g. Colossus, Manchester Baby
  - Terminal-Server and Batch Processing
  - Keyboard, Screen with local machine
  - Cloud, devices, clusters

Pictorial History of Programming UI

- Trend towards higher-level languages and better interfaces/ways to solve problems:
  - Early machine e.g. Colossus, Manchester Baby
  - Terminal-Server and Batch Processing
  - Keyboard, Screen with local machine
  - Cloud, devices, clusters
- As problems have become larger, so the programming models used have adapted to solve these problems

Pictorial History of Programming UI

- Trend towards higher-level languages and better interfaces/ways to solve problems:
  - Early machine e.g. Colossus, Manchester Baby
  - Terminal-Server and Batch Processing
  - Keyboard, Screen with local machine
  - Cloud, devices, clusters
- As problems have become larger, so the programming models used have adapted to solve these problems
- How many lines of code do you think Microsoft Windows has?

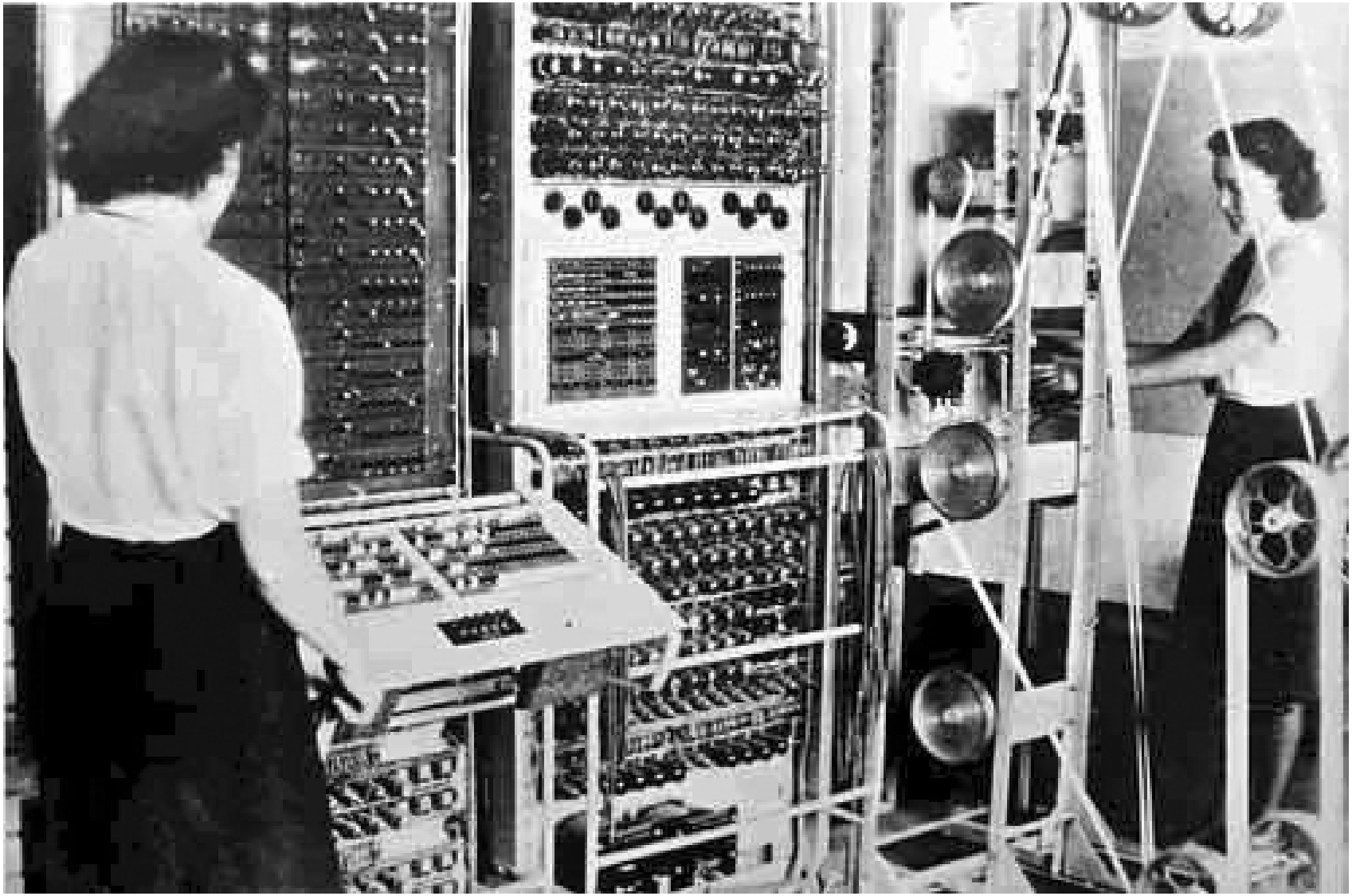
Pictorial History of Programming UI

- Trend towards higher-level languages and better interfaces/ways to solve problems:
  - Early machine e.g. Colossus, Manchester Baby
  - Terminal-Server and Batch Processing
  - Keyboard, Screen with local machine
  - Cloud, devices, clusters
- As problems have become larger, so the programming models used have adapted to solve these problems
- How many lines of code do you think Microsoft Windows has?
  - Windows 7 (2009) reportedly had 40 million lines of code

Pictorial History of Programming UI

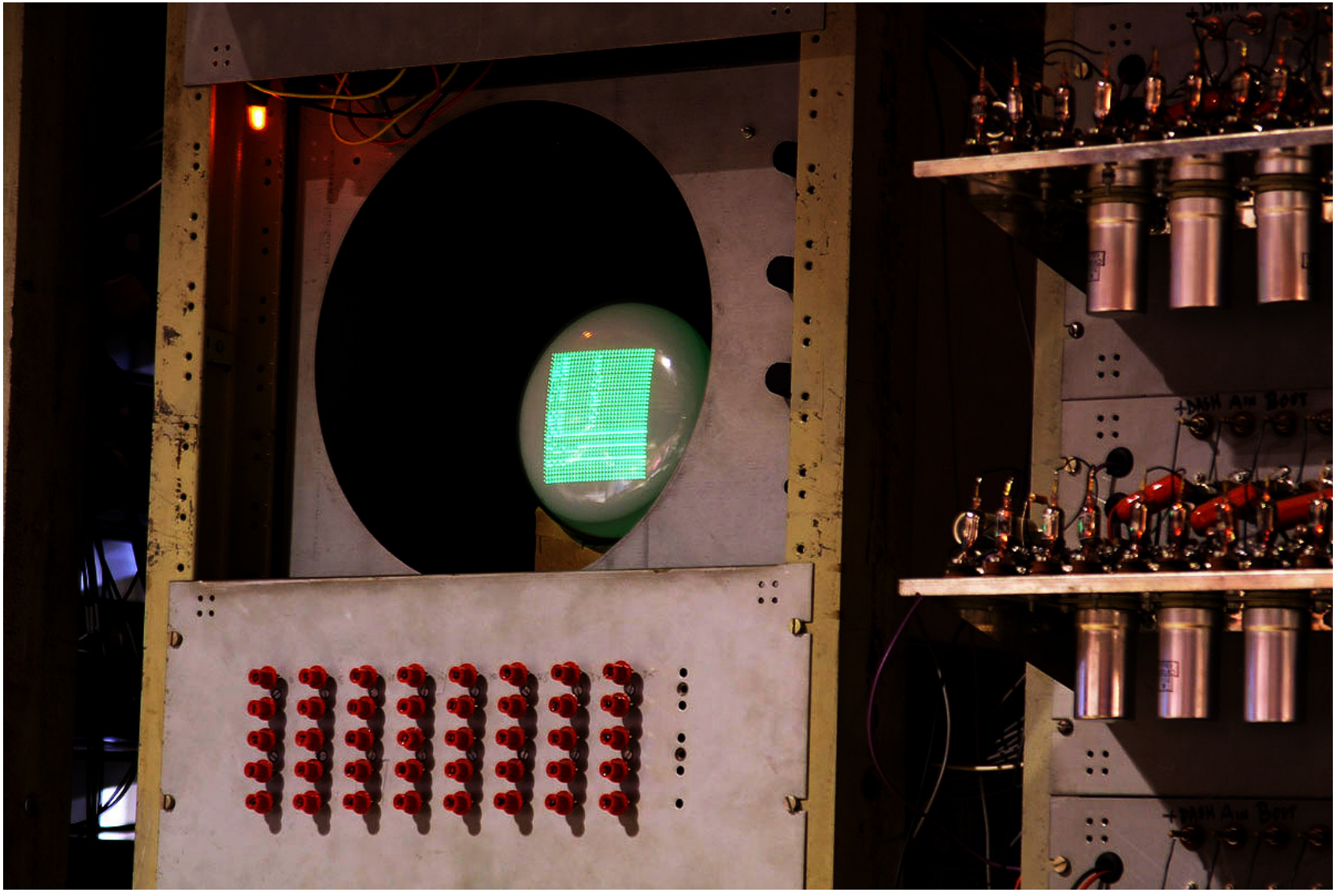
- Trend towards higher-level languages and better interfaces/ways to solve problems:
  - Early machine e.g. Colossus, Manchester Baby
  - Terminal-Server and Batch Processing
  - Keyboard, Screen with local machine
  - Cloud, devices, clusters
- As problems have become larger, so the programming models used have adapted to solve these problems
- How many lines of code do you think Microsoft Windows has?
  - Windows 7 (2009) reportedly had 40 million lines of code
  - A Boeing 787 reportedly has 14 million lines of code

Colossus



Taken from Wikipedia

The Manchester Baby



Taken from Wikipedia

Pictorial History of Programming UI

- Early computer required the use of switches and wires to program

Pictorial History of Programming UI

- Early computer required the use of switches and wires to program
- Computers were generally built for specific purposes

Pictorial History of Programming UI

- Early computer required the use of switches and wires to program
- Computers were generally built for specific purposes
    - Colossus and others for cryptoanalysis

Pictorial History of Programming UI

- Early computer required the use of switches and wires to program
- Computers were generally built for specific purposes
  - Colossus and others for cryptoanalysis
- The “Manchester Baby” had only three programs written for it

Pictorial History of Programming UI

- Early computer required the use of switches and wires to program
- Computers were generally built for specific purposes
  - Colossus and others for cryptoanalysis
- The "Manchester Baby" had only three programs written for it
  - The first two were to find the highest proper factor of 2^{18}

Pictorial History of Programming UI

- Early computer required the use of switches and wires to program
- Computers were generally built for specific purposes
  - Colossus and others for cryptoanalysis
- The "Manchester Baby" had only three programs written for it
  - The first two were to find the highest proper factor of 2^{18}
  - The third was by Alan Turing to perform long division

Pictorial History of Programming UI

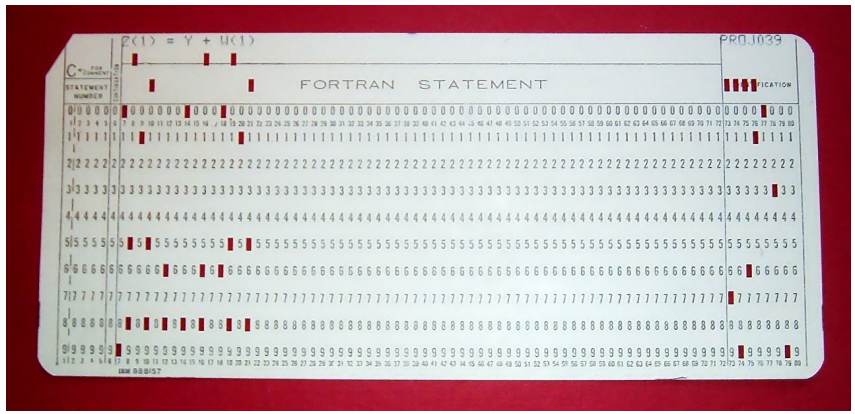
- Early computer required the use of switches and wires to program
- Computers were generally built for specific purposes
  - Colossus and others for cryptoanalysis
- The "Manchester Baby" had only three programs written for it
  - The first two were to find the highest proper factor of 2^{18}
  - The third was by Alan Turing to perform long division
- The Manchester Baby only had eight possible machine instructions

PDP-10

Taken from Columbia University `http://www.columbia.edu/cu/computinghistory/pdp10.html`

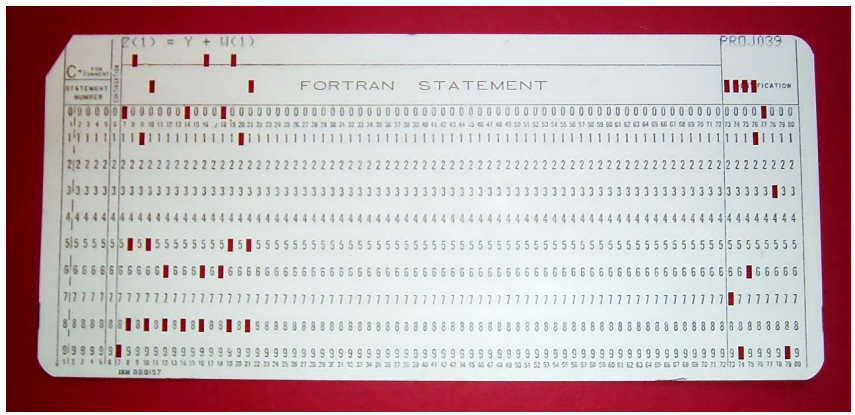
Pictorial History of Programming UI

- Many machines relied on punch cards for programming



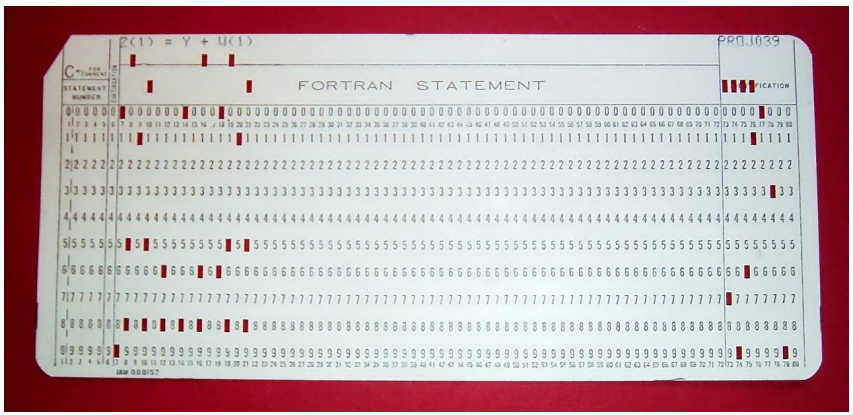
Pictorial History of Programming UI

- Many machines relied on punch cards for programming
- In universities, students would create the punch cards for their program and then have them sent to a computer somewhere for running



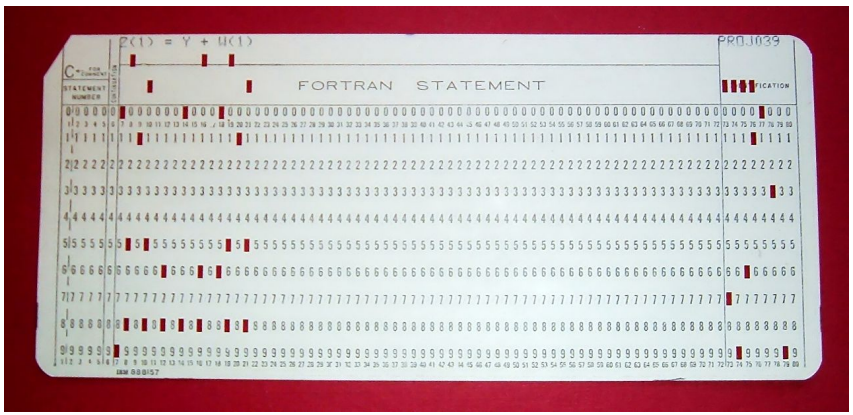
Pictorial History of Programming UI

- Many machines relied on punch cards for programming
- In universities, students would create the punch cards for their program and then have them sent to a computer somewhere for running
- This meant a compile and execute time of days



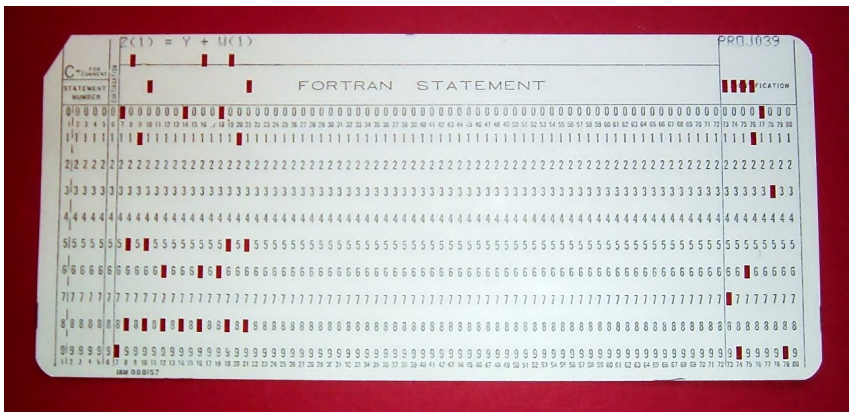
Pictorial History of Programming UI

- Many machines relied on punch cards for programming
- In universities, students would create the punch cards for their program and then have them sent to a computer somewhere for running
- This meant a compile and execute time of days
  - You basically had to ensure no errors before sending it



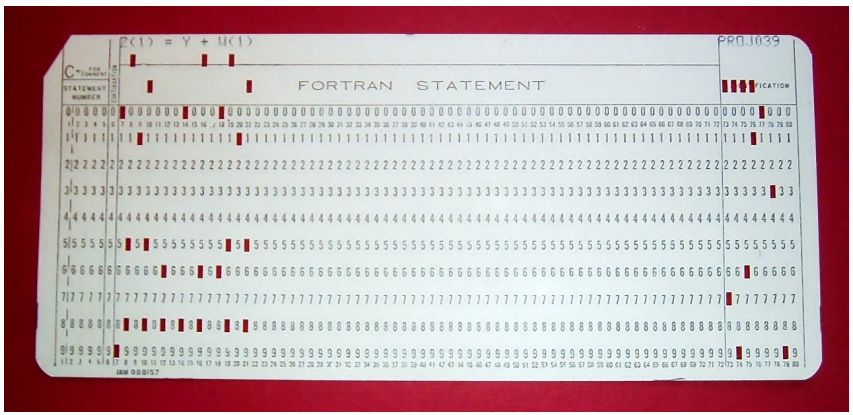
Pictorial History of Programming UI

- Many machines relied on punch cards for programming
- In universities, students would create the punch cards for their program and then have them sent to a computer somewhere for running
- This meant a compile and execute time of days
  - You basically had to ensure no errors before sending it
  - Things are much quicker nowadays, but it normally means novice programmers are more careless



Pictorial History of Programming UI

- Many machines relied on punch cards for programming
- In universities, students would create the punch cards for their program and then have them sent to a computer somewhere for running
- This meant a compile and execute time of days
  - You basically had to ensure no errors before sending it
  - Things are much quicker nowadays, but it normally means novice programmers are more careless
- Punch cards are actually from looms, and were introduced in 1725



Questions?

Outline

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems
- A possible list of classifications is:

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems
- A possible list of classifications is:
  - Procedural (or imperative)

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems
- A possible list of classifications is:
  - Procedural (or imperative)
  - Functional (or declarative)

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems
- A possible list of classifications is:
  - Procedural (or imperative)
  - Functional (or declarative)
  - Logical (also declarative)

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems
- A possible list of classifications is:
  - Procedural (or imperative)
  - Functional (or declarative)
  - Logical (also declarative)
  - Object-Oriented (this week's topic)

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems
- A possible list of classifications is:
  - Procedural (or imperative)
  - Functional (or declarative)
  - Logical (also declarative)
  - Object-Oriented (this week's topic)
- This is just one way we can classify languages.

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems
- A possible list of classifications is:
  - Procedural (or imperative)
  - Functional (or declarative)
  - Logical (also declarative)
  - Object-Oriented (this week's topic)
- This is just one way we can classify languages.
  - Imperative and declarative are broader terms (object-orientation is also imperative)

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems
- A possible list of classifications is:
  - Procedural (or imperative)
  - Functional (or declarative)
  - Logical (also declarative)
  - Object-Oriented (this week's topic)
- This is just one way we can classify languages.
  - Imperative and declarative are broader terms (object-orientation is also imperative)
  - Some languages blur these definitions

Programming Paradigms

- Programming languages can be broken down into a number of *"paradigms"*
  - We can classify a language by the approach it takes to solving problems
- A possible list of classifications is:
  - Procedural (or imperative)
  - Functional (or declarative)
  - Logical (also declarative)
  - Object-Oriented (this week's topic)
- This is just one way we can classify languages.
  - Imperative and declarative are broader terms (object-orientation is also imperative)
  - Some languages blur these definitions
- You can spend a long time arguing with a group of programmers on what a language is

Procedural

- Possibly the only paradigm you are familiar with

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem
  - Essentially thinking like the computer does

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem
  - Essentially thinking like the computer does
- Procedures have a number of different names

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem
  - Essentially thinking like the computer does
- Procedures have a number of different names
  - Routes

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem
  - Essentially thinking like the computer does
- Procedures have a number of different names
  - Routes
  - Subroutines

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem
  - Essentially thinking like the computer does
- Procedures have a number of different names
  - Routes
  - Subroutines
  - Operations

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem
  - Essentially thinking like the computer does
- Procedures have a number of different names
  - Routes
  - Subroutines
  - Operations
  - Methods

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem
  - Essentially thinking like the computer does
- Procedures have a number of different names
  - Routes
  - Subroutines
  - Operations
  - Methods
  - Functions (but don't confuse with mathematical functions or functional programming)

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem
  - Essentially thinking like the computer does
- Procedures have a number of different names
  - Routes
  - Subroutines
  - Operations
  - Methods
  - Functions (but don't confuse with mathematical functions or functional programming)
- Procedures are just a series of steps that must be carried out

Procedural

- Possibly the only paradigm you are familiar with
  - It is the approach we have been taking with C (a procedural language) and C++ (an all sorts of things language)
- Procedural languages describe, step-by-step, the *procedure* that must be followed to solve the problem
  - Essentially thinking like the computer does
- Procedures have a number of different names
  - Routes
  - Subroutines
  - Operations
  - Methods
  - Functions (but don't confuse with mathematical functions or functional programming)
- Procedures are just a series of steps that must be carried out
- These steps are instructions from the programming language

Functional

- As close as we get to doing maths with computers

Functional

- As close as we get to doing maths with computers
- A sequence of *stateless* function evaluations

Functional

- As close as we get to doing maths with computers
- A sequence of *stateless* function evaluations
- A function has inputs/arguments and an output

Functional

- As close as we get to doing maths with computers
- A sequence of *stateless* function evaluations
- A function has inputs/arguments and an output
- Functions can be chained together (e.g. the output from one becomes the input to another)

Functional

- As close as we get to doing maths with computers
- A sequence of *stateless* function evaluations
- A function has inputs/arguments and an output
- Functions can be chained together (e.g. the output from one becomes the input to another)
- Values are assigned and changing them is discourages (pure functional means no mutable variables)

Functional

- As close as we get to doing maths with computers
- A sequence of *stateless* function evaluations
- A function has inputs/arguments and an output
- Functions can be chained together (e.g. the output from one becomes the input to another)
- Values are assigned and changing them is discourages (pure functional means no mutable variables)
- No side-effects. Given the same inputs will always provide the same outputs

Functional

- As close as we get to doing maths with computers
- A sequence of *stateless* function evaluations
- A function has inputs/arguments and an output
- Functions can be chained together (e.g. the output from one becomes the input to another)
- Values are assigned and changing them is discourages (pure functional means no mutable variables)
- No side-effects. Given the same inputs will always provide the same outputs
- Generally heavy on recursion

Functional

- As close as we get to doing maths with computers
- A sequence of *stateless* function evaluations
- A function has inputs/arguments and an output
- Functions can be chained together (e.g. the output from one becomes the input to another)
- Values are assigned and changing them is discourages (pure functional means no mutable variables)
- No side-effects. Given the same inputs will always provide the same outputs
- Generally heavy on recursion
- Has many benefits so functional features are turning up in traditionally non-functional and multi-paradigm languages (that includes C++)

Functional

- As close as we get to doing maths with computers
- A sequence of *stateless* function evaluations
- A function has inputs/arguments and an output
- Functions can be chained together (e.g. the output from one becomes the input to another)
- Values are assigned and changing them is discourages (pure functional means no mutable variables)
- No side-effects. Given the same inputs will always provide the same outputs
- Generally heavy on recursion
- Has many benefits so functional features are turning up in traditionally non-functional and multi-paradigm languages (that includes C++)
- The languages most people have heard of are Lisp, Haskel, Erlang, and F#

Logical

- A paradigm few people ever directly experience

Logical

- A paradigm few people ever directly experience
- Provides automated reasoning over a body of knowledge

Logical

- A paradigm few people ever directly experience
- Provides automated reasoning over a body of knowledge
- It is a type of declarative programming

Logical

- A paradigm few people ever directly experience
- Provides automated reasoning over a body of knowledge
- It is a type of declarative programming
  - Tell the computer what the problem is not how to solve it

Logical

- A paradigm few people ever directly experience
- Provides automated reasoning over a body of knowledge
- It is a type of declarative programming
  - Tell the computer what the problem is not how to solve it
  - For example, build a corpus of facts (about the problem domain) and rules (relating facts with the domain)

Logical

- A paradigm few people ever directly experience
- Provides automated reasoning over a body of knowledge
- It is a type of declarative programming
  - Tell the computer what the problem is not how to solve it
  - For example, build a corpus of facts (about the problem domain) and rules (relating facts with the domain)
  - The facts and rules are logical formulae.

Logical

- A paradigm few people ever directly experience
- Provides automated reasoning over a body of knowledge
- It is a type of declarative programming
  - Tell the computer what the problem is not how to solve it
  - For example, build a corpus of facts (about the problem domain) and rules (relating facts with the domain)
  - The facts and rules are logical formulae.
  - Use logical inference to either derive an answer or to prove the formulae are inconsistent

Logical

- A paradigm few people ever directly experience
- Provides automated reasoning over a body of knowledge
- It is a type of declarative programming
  - Tell the computer what the problem is not how to solve it
  - For example, build a corpus of facts (about the problem domain) and rules (relating facts with the domain)
  - The facts and rules are logical formulae.
  - Use logical inference to either derive an answer or to prove the formulae are inconsistent
- The most widely known logical paradigm language is Prolog

Object-Oriented

- The most commonly used paradigm for commercial software

Object-Oriented

- The most commonly used paradigm for commercial software
- Defines a program as a collection of interacting objects

Object-Oriented

- The most commonly used paradigm for commercial software
- Defines a program as a collection of interacting objects
- Objects are a single unit that combines both data and code

Object-Oriented

- The most commonly used paradigm for commercial software
- Defines a program as a collection of interacting objects
- Objects are a single unit that combines both data and code
  - The properties of the object

Object-Oriented

- The most commonly used paradigm for commercial software
- Defines a program as a collection of interacting objects
- Objects are a single unit that combines both data and code
  - The properties of the object
  - The methods for interacting with that data

Object-Oriented

- The most commonly used paradigm for commercial software
- Defines a program as a collection of interacting objects
- Objects are a single unit that combines both data and code
  - The properties of the object
  - The methods for interacting with that data
- In the most basic form it just means changing how we write our code (not really as simple as this)

Object-Oriented

- The most commonly used paradigm for commercial software
- Defines a program as a collection of interacting objects
- Objects are a single unit that combines both data and code
  - The properties of the object
  - The methods for interacting with that data
- In the most basic form it just means changing how we write our code (not really as simple as this)

  Procedural `do_something(data);`

Object-Oriented

- The most commonly used paradigm for commercial software
- Defines a program as a collection of interacting objects
- Objects are a single unit that combines both data and code
  - The properties of the object
  - The methods for interacting with that data
- In the most basic form it just means changing how we write our code (not really as simple as this)

  Procedural `do_something(data);`

  Object-Oriented `data.do_something();`

Object-Oriented

- The most commonly used paradigm for commercial software
- Defines a program as a collection of interacting objects
- Objects are a single unit that combines both data and code
  - The properties of the object
  - The methods for interacting with that data
- In the most basic form it just means changing how we write our code (not really as simple as this)

  Procedural `do_something(data);`
  Object-Oriented `data.do_something();`
- Numerous languages, but the most commonly discussed are Smalltalk, Ruby, Python, C++, Java, and C#

Questions?

Outline

Object-Oriented Programming

- Usually abbreviated to OO or OOP

Object-Oriented Programming

- Usually abbreviated to OO or OOP

  OO Object-Oriented

Object-Oriented Programming

- Usually abbreviated to OO or OOP

  OO Object-Oriented

  OOP Object-Oriented Programming

Object-Oriented Programming

- Usually abbreviated to OO or OOP

  OO Object-Oriented

  OOP Object-Oriented Programming

- A program is defined as a collection of interacting objects

Object-Oriented Programming

- Usually abbreviated to OO or OOP

  OO Object-Oriented

  OOP Object-Oriented Programming

- A program is defined as a collection of interacting objects
- Objects are a single unit that collect together a set of data and the associated methods for manipulating that data

Object-Oriented Programming

- Usually abbreviated to OO or OOP

  OO Object-Oriented

  OOP Object-Oriented Programming

- A program is defined as a collection of interacting objects
- Objects are a single unit that collect together a set of data and the associated methods for manipulating that data
- An object is defined by two properties

Object-Oriented Programming

- Usually abbreviated to OO or OOP

  OO Object-Oriented

  OOP Object-Oriented Programming

- A program is defined as a collection of interacting objects
- Objects are a single unit that collect together a set of data and the associated methods for manipulating that data
- An object is defined by two properties

  State The data values making up the object

Object-Oriented Programming

- Usually abbreviated to OO or OOP

  OO Object-Oriented

  OOP Object-Oriented Programming

- A program is defined as a collection of interacting objects
- Objects are a single unit that collect together a set of data and the associated methods for manipulating that data
- An object is defined by two properties

  State The data values making up the object

  Behaviour The code or methods that operate on that data

Object-Oriented Programming

- Usually abbreviated to OO or OOP

OO	Object-Oriented
OOP	Object-Oriented Programming

- A program is defined as a collection of interacting objects
- Objects are a single unit that collect together a set of data and the associated methods for manipulating that data
- An object is defined by two properties

State	The data values making up the object
Behaviour	The code or methods that operate on that data

- From this point on many programmers will disagree over details (we are an argumentative bunch)

Object-Oriented Programming

- Usually abbreviated to OO or OOP

  OO Object-Oriented

  OOP Object-Oriented Programming

- A program is defined as a collection of interacting objects
- Objects are a single unit that collect together a set of data and the associated methods for manipulating that data
- An object is defined by two properties

  State The data values making up the object

  Behaviour The code or methods that operate on that data

- From this point on many programmers will disagree over details (we are an argumentative bunch)
- This module will take a particular interpretation of object-orientation that is useful at this stage whilst working with C++

Object-Oriented Programming

- Usually abbreviated to OO or OOP

OO	Object-Oriented
OOP	Object-Oriented Programming

- A program is defined as a collection of interacting objects
- Objects are a single unit that collect together a set of data and the associated methods for manipulating that data
- An object is defined by two properties

State	The data values making up the object
Behaviour	The code or methods that operate on that data

- From this point on many programmers will disagree over details (we are an argumentative bunch)
- This module will take a particular interpretation of object-orientation that is useful at this stage whilst working with C++
  - As you develop as a programmer you will likely adjust your viewpoint of object-orientation

Encapsulation

- Object = data + methods

Encapsulation

- Object = data + methods
- Data is referred to as *attributes* (or member variables, members, properties) of the object

Encapsulation

- Object = data + methods
- Data is referred to as *attributes* (or member variables, members, properties) of the object
- Scope has now been extended. We know have three levels of scope

Encapsulation

- Object = data + methods
- Data is referred to as *attributes* (or member variables, members, properties) of the object
- Scope has now been extended. We know have three levels of scope

  global visible anywhere

Encapsulation

- Object = data + methods
- Data is referred to as *attributes* (or member variables, members, properties) of the object
- Scope has now been extended. We know have three levels of scope

  global visible anywhere

  local visible only in local function/block

Encapsulation

- Object = data + methods
- Data is referred to as *attributes* (or member variables, members, properties) of the object
- Scope has now been extended. We know have three levels of scope

global	visible anywhere
local	visible only in local function/block
object	which can be divided further

Encapsulation

- Object = data + methods
- Data is referred to as *attributes* (or member variables, members, properties) of the object
- Scope has now been extended. We know have three levels of scope

  global visible anywhere

  local visible only in local function/block

  object which can be divided further

  public visible externally from the object

Encapsulation

- Object = data + methods
- Data is referred to as *attributes* (or member variables, members, properties) of the object
- Scope has now been extended. We know have three levels of scope

  global visible anywhere

  local visible only in local function/block

  object which can be divided further

    public visible externally from the object

    private only visible to the object (or objects of the same type)

Encapsulation

- Object = data + methods
- Data is referred to as *attributes* (or member variables, members, properties) of the object
- Scope has now been extended. We know have three levels of scope

  global visible anywhere

  local visible only in local function/block

  object which can be divided further

    public visible externally from the object

    private only visible to the object (or objects of the same type)

    protected visible to the object and objects of child types (more on this soon)

Encapsulation

- Object = data + methods
- Data is referred to as *attributes* (or member variables, members, properties) of the object
- Scope has now been extended. We know have three levels of scope

  global visible anywhere

  local visible only in local function/block

  object which can be divided further

  - **public** visible externally from the object
  - **private** only visible to the object (or objects of the same type)
  - **protected** visible to the object and objects of child types (more on this soon)

- You need to keep track of your scope! In pure object-orientation, there is very little global and local state. Normally you have the object's state

Encapsulation

- So data in an object may be public, private, or protected

Encapsulation

- So data in an object may be public, private, or protected
  - In practice make it private or protected unless you have a good reason to make it public

Encapsulation

- So data in an object may be public, private, or protected
  - In practice make it private or protected unless you have a good reason to make it public
- To do anything with the data (access, modify) requires methods

Encapsulation

- So data in an object may be public, private, or protected
  - In practice make it private or protected unless you have a good reason to make it public
- To do anything with the data (access, modify) requires methods
- Methods provide mechanisms to access data and modify it

Encapsulation

- So data in an object may be public, private, or protected
  - In practice make it private or protected unless you have a good reason to make it public
- To do anything with the data (access, modify) requires methods
- Methods provide mechanisms to access data and modify it
- Abstraction - *only* provide essential information to the outside world and hide the internals of the object

Encapsulation

- So data in an object may be public, private, or protected
  - In practice make it private or protected unless you have a good reason to make it public
- To do anything with the data (access, modify) requires methods
- Methods provide mechanisms to access data and modify it
- Abstraction - *only* provide essential information to the outside world and hide the internals of the object
  - For example, a car provides RPM and speed values externally but hides the internal mechanisms that manipulate these values

Encapsulation

- So data in an object may be public, private, or protected
  - In practice make it private or protected unless you have a good reason to make it public
- To do anything with the data (access, modify) requires methods
- Methods provide mechanisms to access data and modify it
- Abstraction - *only* provide essential information to the outside world and hide the internals of the object
  - For example, a car provides RPM and speed values externally but hides the internal mechanisms that manipulate these values
- *Information Hiding* - segregation of design decisions that may change through the development of the object

Encapsulation

- So data in an object may be public, private, or protected
  - In practice make it private or protected unless you have a good reason to make it public
- To do anything with the data (access, modify) requires methods
- Methods provide mechanisms to access data and modify it
- Abstraction - *only* provide essential information to the outside world and hide the internals of the object
  - For example, a car provides RPM and speed values externally but hides the internal mechanisms that manipulate these values
- *Information Hiding* - segregation of design decisions that may change through the development of the object
  - If access to data is *mediated* by methods then the internal representation of an object can be altered without altering the rest of the program

Encapsulation

- So data in an object may be public, private, or protected
  - In practice make it private or protected unless you have a good reason to make it public
- To do anything with the data (access, modify) requires methods
- Methods provide mechanisms to access data and modify it
- Abstraction - *only* provide essential information to the outside world and hide the internals of the object
  - For example, a car provides RPM and speed values externally but hides the internal mechanisms that manipulate these values
- *Information Hiding* - segregation of design decisions that may change through the development of the object
  - If access to data is *mediated* by methods then the internal representation of an object can be altered without altering the rest of the program
  - So a class can evolve internally over time to better suit its design, but without necessarily affecting the rest of the program

Objects and Classes

- Most popular OO languages are class-based

Objects and Classes

- Most popular OO languages are class-based
  - This means that we define classes

Objects and Classes

- Most popular OO languages are class-based
  - This means that we define classes
  - An object is just an instance of a class

Objects and Classes

- Most popular OO languages are class-based
  - This means that we define classes
  - An object is just an instance of a class
  - The type of object is determined by the type of the class

Objects and Classes

- Most popular OO languages are class-based
  - This means that we define classes
  - An object is just an instance of a class
  - The type of object is determined by the type of the class
  - This is *very* similar to the ideas we looked at with `struct`

Objects and Classes

- Most popular OO languages are class-based
  - This means that we define classes
  - An object is just an instance of a class
  - The type of object is determined by the type of the class
  - This is *very* similar to the ideas we looked at with `struct`
  - The type of the class is basically the interface (attributes and methods)

Objects and Classes

- Most popular OO languages are class-based
  - This means that we define classes
  - An object is just an instance of a class
  - The type of object is determined by the type of the class
  - This is *very* similar to the ideas we looked at with `struct`
  - The type of the class is basically the interface (attributes and methods)
- Classes?

Objects and Classes

- Most popular OO languages are class-based
  - This means that we define classes
  - An object is just an instance of a class
  - The type of object is determined by the type of the class
  - This is *very* similar to the ideas we looked at with `struct`
  - The type of the class is basically the interface (attributes and methods)
- Classes?
  - Extensible "templates" for describing what data and methods an object should support

Objects and Classes

- Most popular OO languages are class-based
  - This means that we define classes
  - An object is just an instance of a class
  - The type of object is determined by the type of the class
  - This is *very* similar to the ideas we looked at with `struct`
  - The type of the class is basically the interface (attributes and methods)
- Classes?
  - Extensible "templates" for describing what data and methods an object should support
  - An object is called an *instance* of a class

- Object = data + methods (yes we are repeating ourselves)

Object-Oriented Analysis and Design

- Object = data + methods (yes we are repeating ourselves)
- OO Analysis & Design (OOAD) is about:

Object-Oriented Analysis and Design

- Object = data + methods (yes we are repeating ourselves)
- OO Analysis & Design (OOAD) is about:
  - Working out which data and methods to include in an object

Object-Oriented Analysis and Design

- Object = data + methods (yes we are repeating ourselves)
- OO Analysis & Design (OOAD) is about:
  - Working out which data and methods to include in an object
  - Ensuring that the program (collection of objects) can address the given problem

Object-Oriented Analysis and Design

- Object = data + methods (yes we are repeating ourselves)
- OO Analysis & Design (OOAD) is about:
  - Working out which data and methods to include in an object
  - Ensuring that the program (collection of objects) can address the given problem
- Objects often "model" the real world and represent real-world things

Object-Oriented Analysis and Design

- Object = data + methods (yes we are repeating ourselves)
- OO Analysis & Design (OOAD) is about:
  - Working out which data and methods to include in an object
  - Ensuring that the program (collection of objects) can address the given problem
- Objects often "model" the real world and represent real-world things
- Designed using a class hierarchy, usually from most general, high-level class descending to the most specific lower-level class

Object-Oriented Analysis and Design

- Object = data + methods (yes we are repeating ourselves)
- OO Analysis & Design (OOAD) is about:
  - Working out which data and methods to include in an object
  - Ensuring that the program (collection of objects) can address the given problem
- Objects often "model" the real world and represent real-world things
- Designed using a class hierarchy, usually from most general, high-level class descending to the most specific lower-level class
  - Composition, inheritance, polymorphism

Object-Oriented Analysis and Design

- Object = data + methods (yes we are repeating ourselves)
- OO Analysis & Design (OOAD) is about:
  - Working out which data and methods to include in an object
  - Ensuring that the program (collection of objects) can address the given problem
- Objects often "model" the real world and represent real-world things
- Designed using a class hierarchy, usually from most general, high-level class descending to the most specific lower-level class
  - Composition, inheritance, polymorphism
- Idea is to turn programming from 'something purely for computer geeks' into 'something for domain experts'

Object-Oriented Analysis and Design

- Object = data + methods (yes we are repeating ourselves)
- OO Analysis & Design (OOAD) is about:
  - Working out which data and methods to include in an object
  - Ensuring that the program (collection of objects) can address the given problem
- Objects often "model" the real world and represent real-world things
- Designed using a class hierarchy, usually from most general, high-level class descending to the most specific lower-level class
  - Composition, inheritance, polymorphism
- Idea is to turn programming from 'something purely for computer geeks' into 'something for domain experts'
  - Instead of working with basic types (`int`, `char`, `string`, etc.) work with objects from the problem domain

Object-Oriented Analysis and Design

- Object = data + methods (yes we are repeating ourselves)
- OO Analysis & Design (OOAD) is about:
  - Working out which data and methods to include in an object
  - Ensuring that the program (collection of objects) can address the given problem
- Objects often "model" the real world and represent real-world things
- Designed using a class hierarchy, usually from most general, high-level class descending to the most specific lower-level class
  - Composition, inheritance, polymorphism
- Idea is to turn programming from 'something purely for computer geeks' into 'something for domain experts'
  - Instead of working with basic types (`int`, `char`, `string`, etc.) work with objects from the problem domain
  - Related to the idea of *Domain Specific Languages*

Object Composition

- Objects can contain attributes which are other other objects

Object Composition

- Objects can contain attributes which are other other objects
  - A car contains objects such as a wheel (4 of them), engine, exhaust, etc.

Object Composition

- Objects can contain attributes which are other other objects
  - A car contains objects such as a wheel (4 of them), engine, exhaust, etc.
- So an object can be composed from other objects

Object Composition

- Objects can contain attributes which are other other objects
  - A car contains objects such as a wheel (4 of them), engine, exhaust, etc.
- So an object can be composed from other objects
- Useful when modelling real-world hierarchies:

Object Composition

- Objects can contain attributes which are other other objects
  - A car contains objects such as a wheel (4 of them), engine, exhaust, etc.
- So an object can be composed from other objects
- Useful when modelling real-world hierarchies:
  - A car is defined in terms of sub-objects representing the wheels, engine, chassis

Object Composition

- Objects can contain attributes which are other other objects
  - A car contains objects such as a wheel (4 of them), engine, exhaust, etc.
- So an object can be composed from other objects
- Useful when modelling real-world hierarchies:
  - A car is defined in terms of sub-objects representing the wheels, engine, chassis
  - These sub-objects may also be made up of sub-objects (e.g. piston)

Object Composition

- Objects can contain attributes which are other other objects
  - A car contains objects such as a wheel (4 of them), engine, exhaust, etc.
- So an object can be composed from other objects
- Useful when modelling real-world hierarchies:
  - A car is defined in terms of sub-objects representing the wheels, engine, chassis
  - These sub-objects may also be made up of sub-objects (e.g. piston)
  - Enables abstraction (encapsulation, data hiding) to vary within the program to match the level at which the object is being used

Object Composition

- Objects can contain attributes which are other other objects
  - A car contains objects such as a wheel (4 of them), engine, exhaust, etc.
- So an object can be composed from other objects
- Useful when modelling real-world hierarchies:
  - A car is defined in terms of sub-objects representing the wheels, engine, chassis
  - These sub-objects may also be made up of sub-objects (e.g. piston)
  - Enables abstraction (encapsulation, data hiding) to vary within the program to match the level at which the object is being used
- Implements a **has-a** type of relationship between objects

Inheritance

- When an object is based on, or is a more specialised version of, a more abstract/higher-level object

Inheritance

- When an object is based on, or is a more specialised version of, a more abstract/higher-level object
- The relationship between the parent (or base) class and child (or derived) class is the **is-a** relationship

Inheritance

- When an object is based on, or is a more specialised version of, a more abstract/higher-level object
- The relationship between the parent (or base) class and child (or derived) class is the **is-a** relationship
  - e.g. vehicle as a base class

Inheritance

- When an object is based on, or is a more specialised version of, a more abstract/higher-level object
- The relationship between the parent (or base) class and child (or derived) class is the **is-a** relationship
  - e.g. vehicle as a base class
  - car, bus, lorry, are vehicles. They are child classes of a vehicle

- **Runtime polymorphism**

- **Runtime polymorphism**
  - Using a pointer to a based class (or a reference) to manipulate derived classes

- **Runtime polymorphism**
  - Using a pointer to a based class (or a reference) to manipulate derived classes
  - For example, classes for moggy, tiger, and ocelot all inheriting from the cat class and all overriding the meow method

- **Runtime polymorphism**
  - Using a pointer to a based class (or a reference) to manipulate derived classes
  - For example, classes for moggy, tiger, and ocelot all inheriting from the cat class and all overriding the meow method
    - Since all inherit from cat and all implement meow we call meow on any object but have a pointer to a moggy, tiger, or ocelot class

- **Runtime polymorphism**
  - Using a pointer to a based class (or a reference) to manipulate derived classes
  - For example, classes for moggy, tiger, and ocelot all inheriting from the cat class and all overriding the meow method
    - Since all inherit from cat and all implement meow we call meow on any object but have a pointer to a moggy, tiger, or ocelot class
    - The correct version of the method is called

- **Runtime polymorphism**
  - Using a pointer to a based class (or a reference) to manipulate derived classes
  - For example, classes for moggy, tiger, and ocelot all inheriting from the cat class and all overriding the meow method
    - Since all inherit from cat and all implement meow we call meow on any object but have a pointer to a moggy, tiger, or ocelot class
    - The correct version of the method is called
  - We have already used coercion polymorphism (casting) back at the start of the module

- **Runtime polymorphism**
  - Using a pointer to a based class (or a reference) to manipulate derived classes
  - For example, classes for moggy, tiger, and ocelot all inheriting from the cat class and all overriding the meow method
    - Since all inherit from cat and all implement meow we call meow on any object but have a pointer to a moggy, tiger, or ocelot class
    - The correct version of the method is called
  - We have already used coercion polymorphism (casting) back at the start of the module
  - We will look at this further in the workbook. However, this becomes more important later in your programming education

Bonus Agent Oriented Programming

- An object does something because you tell it to:

Bonus Agent Oriented Programming

- An object does something because you tell it to:
  - Make call to a public method and the object has to execute it

Bonus Agent Oriented Programming

- An object does something because you tell it to:
  - Make call to a public method and the object has to execute it
- An agent does something because it wants to

Bonus Agent Oriented Programming

- An object does something because you tell it to:
  - Make call to a public method and the object has to execute it
- An agent does something because it wants to
  - Send message to agent and agent evaluates whether the message aligns with its own goals

Bonus Agent Oriented Programming

- An object does something because you tell it to:
  - Make call to a public method and the object has to execute it
- An agent does something because it wants to
  - Send message to agent and agent evaluates whether the message aligns with its own goals
  - **Note** might also be motivated to perform an action waiting to be asked

Questions?

Outline

Summary

- We covered a bit of history of programming user interfaces

Summary

- We covered a bit of history of programming user interfaces
  - Aren't you glad of the tools you have now!

Summary

- We covered a bit of history of programming user interfaces
  - Aren't you glad of the tools you have now!
- We also looked at the different paradigms programming languages take

Summary

- We covered a bit of history of programming user interfaces
  - Aren't you glad of the tools you have now!
- We also looked at the different paradigms programming languages take
- Our main focus has been on object-orientation

Summary

- We covered a bit of history of programming user interfaces
  - Aren't you glad of the tools you have now!
- We also looked at the different paradigms programming languages take
- Our main focus has been on object-orientation
  - Data and methods

Summary

- We covered a bit of history of programming user interfaces
  - Aren't you glad of the tools you have now!
- We also looked at the different paradigms programming languages take
- Our main focus has been on object-orientation
  - Data and methods
  - Encapsulation

Summary

- We covered a bit of history of programming user interfaces
  - Aren't you glad of the tools you have now!
- We also looked at the different paradigms programming languages take
- Our main focus has been on object-orientation
  - Data and methods
  - Encapsulation
  - Composition

Summary

- We covered a bit of history of programming user interfaces
  - Aren't you glad of the tools you have now!
- We also looked at the different paradigms programming languages take
- Our main focus has been on object-orientation
  - Data and methods
  - Encapsulation
  - Composition
  - Inheritance

Summary

- We covered a bit of history of programming user interfaces
  - Aren't you glad of the tools you have now!
- We also looked at the different paradigms programming languages take
- Our main focus has been on object-orientation
  - Data and methods
  - Encapsulation
  - Composition
  - Inheritance
  - Polymorphism

To do...

- For further reading (from the very basics of how computer circuitry is developed) see *Code: The Hidden Language of Computer Hardware and Software*, by Charles Petzold

To do...

- For further reading (from the very basics of how computer circuitry is developed) see *Code: The Hidden Language of Computer Hardware and Software*, by Charles Petzold
  - This is an interesting read aimed not at more than just computing people

To do...

- For further reading (from the very basics of how computer circuitry is developed) see *Code: The Hidden Language of Computer Hardware and Software*, by Charles Petzold
  - This is an interesting read aimed not at more than just computing people
- In the lab - object oriented programming with C++.

To do...

- For further reading (from the very basics of how computer circuitry is developed) see *Code: The Hidden Language of Computer Hardware and Software*, by Charles Petzold
  - This is an interesting read aimed not at more than just computing people
- In the lab - object oriented programming with C++.
  - Feel free to continue using Visual Studio if you want. However, coursework 2 submission will be code and make files again.

To do...

- For further reading (from the very basics of how computer circuitry is developed) see *Code: The Hidden Language of Computer Hardware and Software*, by Charles Petzold
  - This is an interesting read aimed not at more than just computing people
- In the lab - object oriented programming with C++.
  - Feel free to continue using Visual Studio if you want. However, coursework 2 submission will be code and make files again.
- Coursework 2 will appear on Moodle today (doing some final tweaks).

To do...

- For further reading (from the very basics of how computer circuitry is developed) see *Code: The Hidden Language of Computer Hardware and Software*, by Charles Petzold
  - This is an interesting read aimed not at more than just computing people
- In the lab - object oriented programming with C++.
  - Feel free to continue using Visual Studio if you want. However, coursework 2 submission will be code and make files again.
- Coursework 2 will appear on Moodle today (doing some final tweaks).
- Next week - virtual behaviour and polymorphism.