Introduction to Programming
COMP07027

Introduction

Module Co-ordinator: Dr Richard Beeby
Room E258, E Building South, Paisley
Richard.Beeby@uws.ac.uk

Module Lecturers

- Ayr: Brian McGhee
- Dumfries: Rebecca Redden
- Hamilton: Miriam Birch
- Paisley: Richard Beeby

Module Summary

- This module is an introduction to how to design and develop computer programs.
  - The module does not assume that you have done any programming in the past.
- The programming language we will use is Java (Standard Edition 6 or later).
- The module will involve you writing one or more small programs most weeks.
  - The only way to learn how to program is to write programs!

Teaching Schedule

- The module will run over 24 teaching weeks (that is, over **two** trimesters)
- There will be a **lecture** most weeks.
- In addition, each week there will be a **laboratory** class with exercises
- There will be online quizzes some weeks for you to do
- You will be expected to attend all the classes, take the quizzes, and do the exercises
  - We will take attendance at classes
  - We will ask you to submit a few of the exercises online so we can see how you are doing and that you are engaging with the module

Resources Used in the Module

- Moodle site for the module at http://moodle.uws.ac.uk
  - Lecture slides, tutorial-type questions & lab exercises, quizzes and the programming assignment will be published here, and this is where you will submit work
- Free electronic textbook by David Eck – online and download in Portable Document Format (PDF)
  http://math.hws.edu/javanotes/
- Java Standard Edition 8, free download from www.oracle.com/technetwork/java/javase/downloads/index.html
- Eclipse development environment, free download from http://www.eclipse.org/downloads/

Reading

- David Eck's textbook
  - you will be expected to read material from the textbook each week for the following week's classes
  - will be used for some lab exercises
    - The online version of the book contains a discussion of the solution for each exercise, followed by a solution in Java.
  - includes a class called **TextIO** that will be used for reading input from keyboard and file and writing output to screen and file (always carry a copy of this on your USB drive)
  - also includes material on graphical user interfaces that this module does not cover

If you don't like the textbook, try this one (but its not free!)

- Khalid Azim Mughal, Torill Hamre & Rolf W. Rasmussen (2008). **Java Actually: A Comprehensive Primer in Programming**. Cengage Learning. ISBN: 978-1-844480-933-2
- Website with resources for this book is (link also on Moodle):
  - http://www.ii.uib.no/~khalid/jac/

Module Learning Outcomes – what you will learn to do

- At the end of this module the student will be able to:
  - L1. write small structured programs in a high level programming language
  - L2. demonstrate the use of standard programming constructs for iteration, selection and data structures such as arrays
  - L3. create a simple console-based user-interface for a program and use this to create interactive software

from the module descriptor

Assessment – how you will demonstrate that you have learned to do those things

- Class tests – online quizzes
  - Two of these will be worth 10% of the module marks each.
- Programming project
  - There will be a programming project for you to do in trimester 2 worth 80% of the module marks. You will be able to work in pairs on this.

Attendance and Engagement revisited!

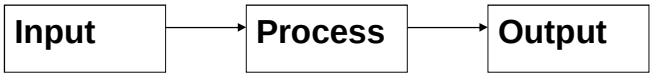
- University policy on Attendance and Engagement
  - To repeat: you are expected to attend all the classes for this module and to attempt the exercises
  - If you are unwell and cannot attend you should contact the School Office and let us know – you'll need to complete a Self-Certification form when you return
    - or if you are absent more than 7 days you'll need a Medical Certificate or equivalent

Introduction to Programming

1: Programming and Structured Programming

Computer Systems

- Can be...
  - Specialised – e.g. MP3 player, Calculator, games console
  - General purpose – e.g. PC, Macintosh
- In all cases, their operation can be generalised as...



Computer Software

- Software is the set of instructions given to a computer
  - Specialised – MP3 Decode/Control, maths functions, games (graphics, sound, simple logic)
  - General purpose – databases, spreadsheets, word processing, etc.

Computer hardware

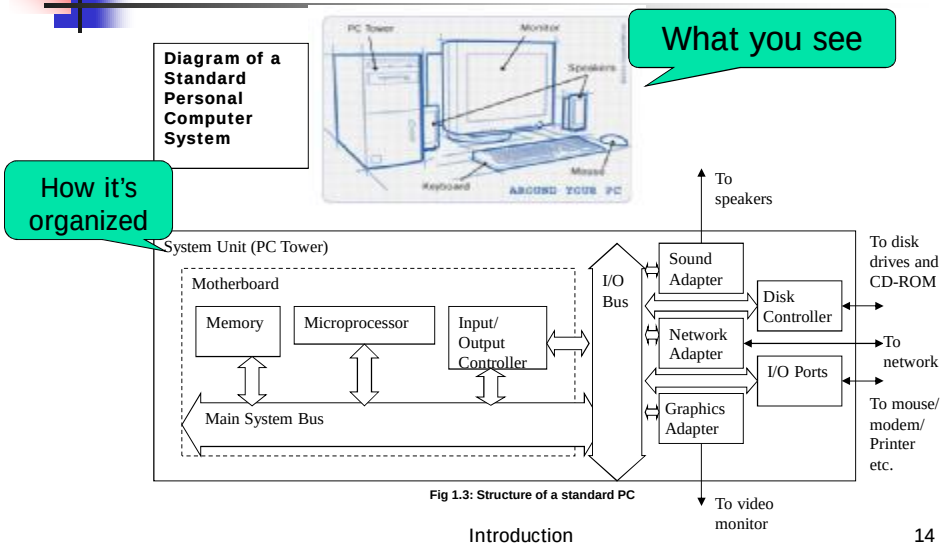


Fig 1.3: Structure of a standard PC

General Purpose Software

- All computers need an *Operating System*
  - Software that controls interactions between the CPU and external hardware
- PC uses Microsoft Windows, Linux, Unix, BeOS (mostly Windows)
  - Provide basic I/O for disc, keyboard, modems, printers, graphics etc.
  - Provide basic *Shell Functions* (command centres)
  - Provide filing system(s)
  - Hosts *Applications Programs* – word processing etc.

Computer Programs

- Computers obey simple instructions
  - Add, subtract and simple arithmetic
  - Compare two values
  - Look up a value in a table
  - Copy a block of data from one part of memory to another
  - Transfer data to and from output devices
- These instructions are called *Machine Code*
  - It takes a lot of machine code to do anything useful
  - This type of code is difficult to write and debug
  - Only very fundamental operations are written in this form

Computer programs II

- Since a computer is good at doing trivial things repetitively, we can use it to translate more functional (or *higher level*) code into machine code
  - Assembly Language (still pretty low level)
  - C/C++, BASIC, FORTRAN, COBOL, Pascal, Ada (compiled high-level languages - HLL)
  - Visual Basic, Access macros, Javascript, PERL, Python (interpreted scripting languages)
- You (the programmer) write the high level code, a Compiler (HLL) or Interpreter (Script) translates it into machine code. Compilers and Interpreters are computer programs.

Computer programs III

- A *compiler* translates the whole program in to machine code. The machine code can then be executed whenever needed.
- An *interpreter* runs the high level program by translating the next statement of the program in to machine code and then executing that machine code. It repeats this till the program terminates.
- Modern language implementations (for example, of Java and Python) combine these two approaches by compiling the whole program in to a low level language (for Java, bytecode) for a *virtual machine.* The bytecode program can be run on an actual computer by an interpreter.

C/C++

- C was developed in the early 1970s as a *systems programming language*
  - Combination of high level and low-level features
  - Speed of machine code with features to support *structured programming* and *modular programming*
  - Small core language made it easily portable to any computer system
- C++ was developed as “a better C” in the early 1980s (name first used in 1983). Added new features, including support for *object-oriented programming*
  - Hugely powerful but large core language
  - Can be highly complex
  - Compatible with C so retains low-level features

Java

- Java version 1.0 development kit released in 1996 – Java developed by Sun “Green Team” 1991-1995.
  - Uses C-style syntax but not compatible with C
  - Small core language
  - Pure and *strongly typed* object-oriented language
  - Design aimed for platform independence (Java virtual machine - JVM) so well suited as a language for the internet at just the time the internet was taking off
- Major revisions and updates to the language in each subsequent version, particularly versions 2, 5 and 8
- This module assumes Java standard edition version 6 or later
  - current version is 8

Input-Process-Output Model

- Any simple program does this
  - Get input data
  - Process it in some way
  - Output the result
- Idea: when designing computer software, we should aim to model any part of a program as an Input-Process-Output subsystem

A simple Java program

```
public class Application {
  public static  float FtoC(float F) {
        return (F-32) * 5 / 9;
  } // end FtoC()

  public static void main(String[] args ) {
        float tempF, tempC;
        TextIO.put("Enter temperature in °F: ");
        tempF = TextIO.getlnFloat();
        tempC = FtoC(tempF);
        TextIO.putln("Temperature in °C is " + tempC);
  } // end main()
} // end Application
```

Java code is always in a class

This method defines a *function* – something that calculates and returns a result

This is the *main method* of the program

This *calls* the defined function

Input

Process

Output

Separating Data and Process

- Items in programs can be put into two broad categories
  - Data – the encoded information a program inputs, calculates, stores, retrieves and outputs
  - Process – the instructions that perform operations on the data
- Two approaches in current use:
  - Keep data and process separate (structured programming)
  - Combine data and process to create active entities in software (object-oriented programming)
- This module covers structured programming, though we will look at some object-based ideas as well

Building Complex Software

- Simple tasks require little or no organisation
  - e.g. Building a garden wall
    - Do calculations on the back of an envelope
- More complex tasks require an organised approach
  - e.g. building a house
    - Planning, coordination, civic requirements etc.
- Highly complex tasks require very detailed organisation
  - e.g. building an office block
    - Large scale communication (between workforces), detailed planning, complex legal frameworks, massive coordination etc.

The Software Development Process

1. Start with a top-level requirements statement – e.g. a program to collate student records
2. Analyse this to determine data requirements, processing requirements, inputs, outputs , assumptions made, user's needs
3. Based on analysis, design the broad structure of the program - what data elements there are, how they will be processed
4. Design the detailed structure of the program components
5. Write the code
6. Test and debug the code

Input and Output

- In simple programs, the user is the source of input (through the keyboard) and the ultimate destination of output (via the screen)
- Java provides a library package called java.io that provides a rich set of facilities for input and output
  - The input facilities are quite complicated for a beginning programming course, however
- TextIO
  - We will use a class called TextIO to simplify input/output
  - TextIO uses classes from java.io and was written by (and accompanies the module textbook by) David Eck
  - To use the class include the code for it in the same folder as your application
  - Use this class for all keyboard input and screen output

Printing to the console (see Eck section 2.4)

- Using TextIO to display text and values on the console window

```
public class HelloWorld {
  public static void main(String[] args) {
     TextIO.putln("Hello World!");
  }
}
```

A String literal

Print and terminate the line

Notes on printing to the console

- When a string is printed, it is displayed without its quotation marks
- `TextIO.putln()` displays the value and then terminates the line, so that output continues on the next line
- `TextIO.put()` displays the value without terminating the line, so output continues on the same line

System.out

- Java provides an object named *System.out* to print to the standard output (by default, the console)
  - System.out.print() does the same as TextIO.put()
  - System.out.println() does the same as TextIO.putln()
  - System.out only does output, TextIO also provides for input from the keyboard
  - The examples in the lectures and the book occasionally use System.out.print() to remind you that this is the standard Java way of printing to the console.

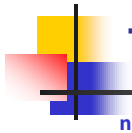
Printing to the console

- To print several things on the same line
  - Can call TextIO.put() for each

```
TextIO.put("Some text ");
TextIO.put("same line as before. ");
TextIO.putln("End of the line.");
```

  - Or can use the String + operator to combine them in one call

```
TextIO.putln("Some text " +
             "same line as before. " +
             "End of the line.");
```

This Module

- In this module we will concentrate on learning to program
- We will use Java
  - Input/output uses class called TextIO supplied by author of module textbook
  - To write structured programs and to look at how to group data with the operations that act on it (object-based programming)
- By the end of the module, aim is...
  - Good structured programming skills
  - Good understanding of programming concepts
  - Ability to develop algorithms
  - Ability to create and use structured data

Reading for Next Week

- Eck, **chapter 1**
  - this is a more extended treatment of what we covered today, and also has some coverage of
    - user interface design
    - the internet
- **Eck, chapter 2**
  - Read at least sections 2.1 and 2.2