Programming Design and Implementation

Lecture 1: Introduction

Dr David A. McMeekin
Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

COMP1007 - Unit Learning Outcomes

- Identify appropriate primitive data types required for the translation of pseudocode algorithms into Java;
- Design in pseudocode simple classes and implement them in Java in a Linux command-line environment;
- Design in pseudocode and implement in Java structured procedural algorithms in a Linux command-line environment;
- Apply design and programming skills to implement known algorithms in real world applications; and
- Reflect on design choices and communicate design and design decisions in a manner appropriate to the audience.

COMP5011 - Unit Learning Outcomes

- Develop and apply simple non-object oriented algorithms;
- Develop and implement simple classes in an object oriented language;
- Create object oriented designs consisting of classes connected by aggregation; and
- Communicate design and design decisions in a manner appropriate to the audience.

Outline

Computer Basics

What is a Computer?

- Machine that accepts data, processes it and produces output
- Computer consists of:
  - CPU
  - Input/output devices
  - Dynamic memory
  - Secondary storage
- So it can run software:
  - Systems software:
    - Operating systems, network tools, software development tools (e.g. compilers)
  - Application software:
    - Word processors, spreadsheets, databases, modelling and simulation.

Software

- A program is a set of instructions to a computer.
- These instructions are written in a programming language.
- Each computer understands one language, its machine language, not (usually) human readable.
- A program is written in a high level language (for humans) and then translated into machine code (for the computer).
- The purpose of you studying this unit is to design and write well structured, robust, maintainable software - not just to write a program that works some of the time.

Programming Languages

- 1940's — Machine language
- Early 50's — Assembly language
- Late 50's — High level languages & compilers introduced
  - Fortran
  - COBOL
- 70's and 80's — Emergence of better structured languages
  - Pascal
  - Ada
  - C
- 90's — Object Oriented Languages
  - C++ *(or was it?)*
  - Java
  - Perl
- 2000's
  - C#
  - Python
  - Ruby

Data Representation in Computer Memory

- Computer memory is made up of components that can be in one of two states (on or off)
- Other binary methods of information include:
  - On and Off (Flashing light)
  - Dot and Dash (Morse)
  - North and South (Magnet)
- Can be used in two ways:
  - Represented actual values in base 2
  - Hold an arbitrary series of states coded with particular meanings

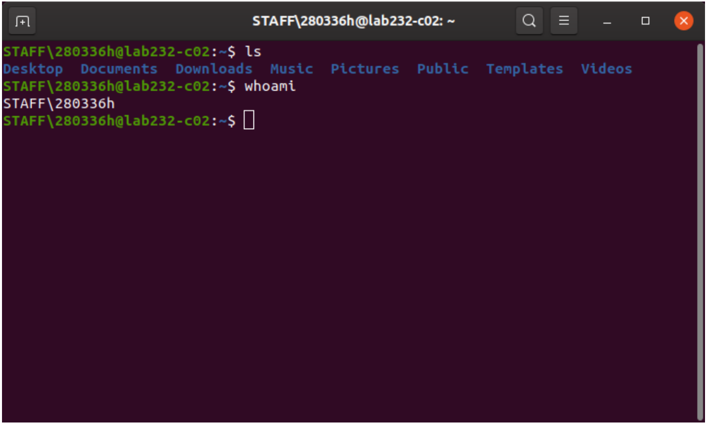
UNIX

Introduction to UNIX

- We will use the Linux operating system
  - Linux is a variant of UNIX
- UNIX is:
  - Totally different to what you might be used to
  - An operating system which provides far greater scope in what can be accomplished but at a cost of a more difficult to learn user interface
- macOS is a UNIX operating system
- As future computing professionals you must be familiar with the UNIX or UNIX-like environment

Command Line Interfaces

- A command line interface is composed of:
  - A prompt which signals the user when they can type commands
  - A command line on which the typed commands appear



CLI vs GUI

- Graphical User Interfaces are:
  - Easy to learn how to use
  - Inefficient when dealing with repeated sets of the same functionality
- Command Line Interfaces are:
  - Require understanding of a number of concepts
  - Require familiarity with a command language
  - Very fast
  - Highly efficient, when dealing with repeated sets of the same functionality
    - e.g., Multiple files that have a similar name or extension
- In PDI we will mostly deal with the Command Line Interface

UNIX Shells

- Under UNIX the Command Line Interface is known as a Shell
- Shells allow us to perform commands on the Operating System as a user
  - Think of it as the communication medium between you and the hardware
- The Shell we will be dealing with in PDI will be `bash`
  - **B**ourne **A**gain **SH**ell
  - This will be covered in depth in Unix and C Programming (COMP1000)

Paths and Directory Structure

- Within UNIX everything is a file, and hence can be accessed with a direct path
- Generally speaking we will only be accessing our files in our Home (∼) directory
  - This means we may use the shortcut 'tilde' to access home from anywhere

```
dam@314lab:~$ cd ~/Documents/PDI
```

- However we may also give a direct path, specified from the root directory (/)

```
dam@314lab:~$ cd /home/mark/Documents/PDI
```

- Any time you specify a path you **must** make sure it is accurate, otherwise the OS does not know where you are talking about

Common Bash (UNIX) Commands

- cd: Change Directory
  - Moves to a different directory, specified after the command

```
dam@314lab:~$ cd Documents
```

- cp: Copy
  - Copies a file or folder from source to destination

```
dam@314lab:~$ cp Source.txt Destination.txt
```

- mv: Move
  - Moves a file or folder from source to destination
    - Note: The original file will not remain

```
dam@314lab:~$ mv Source.txt Destination.txt
```

Common Bash (UNIX) Commands (2)

- `ls`: List
  - Lists all (non hidden) files and folders in the current directory
    - Note: You may specify "`-la`" to display hidden files aswell
    - *This is L (el) not 1 (one)*

```
dam@314lab:~$ ls
```

- `mkdir`: Make Directory
  - Makes a new folder, specified after the command

```
dam@314lab:~$ mkdir PDI_Worksheets
```

- `rm`: Remove
  - Removes a file, specified after the command
    - Note: To remove folders, you need to specify "`-rf`"
    - Be careful! This will remove with force and can not recover it

```
dam@314lab:~$ rm MyFile.txt
```

Live Demo

VIM

VI(M) - visual editor

- A text editor created for UNIX;
- It is primitively powerful, like nothing you've used before;
- Launch from the Terminal window;

```
dam@314lab:~$ vim
```

Why use VIM?

- It is available on all UNIX and Linux systems;
- Allows easy edit access to system files from the Terminal;

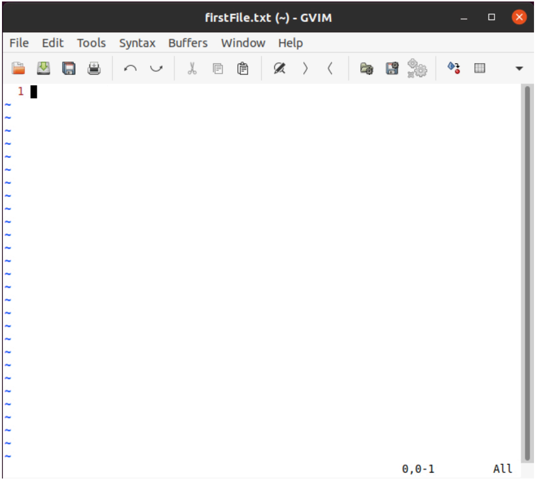
```
dam@314lab:~$ vim .bashrc
```

Terminal

```
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

```

gVIM - VIM with a GUI

```
dam@314lab:~$ gvim firstFile.txt
```



VIM/gVIM has Two Modes

- **Command** - causes actions to be executed on the file;
  - Default/Starting mode;
  - Each letter typed may be a command executed on the file.

- **Insert** - edit the file's content;
  - Change to edit mode by pressing Esc followed by i key;
  - The file can now be edited.

Live Demo of VIM/gVIM

Binary

Data Representation in Computer Memory

- Computer memory is made up of components that can be in one of two states: on or off;
- Other binary methods of information include:
  - On and Off (Flashing light)
  - Dot and Dash (Morse)
  - North and South (Magnet)
- Can be used in two ways:
  - Represented actual values in base 2
  - Hold an arbitrary series of states coded with particular meanings

Terminology

- Each binary digit is called a **bit**
- A group of eight (8) bits is a **byte**
- Memory is broken up into **wordsize** storage locations
- Wordsize: machine dependent and one or more bytes long
  - Now 64 bits: 8 bytes
- Each memory location is located through its **memory address**
- All data and programs are stored in memory using various interpretations of these groups of 1's and 0's

Data Types

- Manner of interpretation of the 1's and 0's varies for different data types stored:
- The interpretation of the 1's and 0's depends on the data type being represented:
  - Address;
  - Instruction;
  - Integer value;
  - Real value;
  - Boolean
  - Characters:
    - Single characters
    - Character Strings

Integer Range

- Determined by how many distinct base2 values can be stored in the given number of bits:
  - every additional bit doubles the size of the range.
- For N bits:
  - 1 bit required for the sign;
  - the remaining N-1 bits can represent 2^{N-1} different combinations, directly related to the binary value
- Note: the lack of symmetry is because of the need to represent zero (0) as one of the 2^{N-1} values:
  - {2^{N-1} negative, 0, 2^{N-1}-1 positive} values
    - Negative values stored as the 2's compliment of the number
- Attempting to store a number larger/smaller than the maximum/minimum value leads to **Integer Overflow.**

Decimal Number System

- We use the base-10, `Decimal`, number system;
- Digits used: `0 - 9`, (`0 1 2 3 4 5 6 7 8 9`)
- 2546:

10^3	10^2	10^1	10^0
1000	100	10	1
2	5	4	6

$$(2\text{x}1000) + (5\text{x}100) + (4\text{x}100) + (6\text{x}1) = 2546$$

Computer Used Number System

- There are 10 types of people:
  1. those who understand binary; and
  2. those who don't.
- Correctly written: there are 10_2 types of people;

- Computers use base-2 (Binary);
- Often represented by base-8 (Octal) or base-16 (Hexadecimal);
- Easy to convert as they fall along the base-2 scale.

Decimal to Binary

- Use the Quotient Remainder method;
- Divide the number by 2 - if there is a remainder, insert a 1, otherwise insert a 0 (starting from the right)

```
2546 ÷ 2 = 1273    | Remainder: 0      // 0
1273 ÷ 2 = 636     | Remainder: 1      // 10
 636 ÷ 2 = 318     | Remainder: 0      // 010
 318 ÷ 2 = 159     | Remainder: 0      // 0010
 159 ÷ 2 = 79      | Remainder: 1      // 10010
  79 ÷ 2 = 39      | Remainder: 1      // 110010
  39 ÷ 2 = 19      | Remainder: 1      // 1110010
  19 ÷ 2 = 9       | Remainder: 1      // 11110010
   9 ÷ 2 = 4       | Remainder: 1      // 111110010
   4 ÷ 2 = 2       | Remainder: 0      // 0111110010
   2 ÷ 2 = 1       | Remainder: 0      // 00111110010
   1 ÷ 2 = 0       | Remainder: 1      // 100111110010
```

Double Check

- 100111110010

2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	0	1	1	1	1	1	0	0	1	0

$$2048 + 0 + 0 + 256 + 128 + 64 + 32 + 16 + 0 + 0 + 2 + 0 = 2546$$

Your Turn

- Convert 126_{10} to a Binary number;
- Convert 1101011_2 to a Decimal number.
- Check your work by converting them back.

Octal & Hex

Octal

- Base 8
- Digits used: 0 - 7 (0 1 2 3 4 5 6 7)
- 4762_8:

8^3	8^2	8^1	8^0
512	64	8	1
4	7	6	2

- Converting from Octal to Decimal (base 10):

$$(4x512) + (7x64) + (6x8) + (2x1) = 2546_{10}$$
$$4762_8 = 2546_{10}$$

Decimal to Octal

- Use the Quotient Remainder method;
- Divide the number by 8 - if there is a remainder, insert remainder, (starting from the right):

```
2546 ÷ 8 = 318    | Remainder: 2    // 2
 318 ÷ 8 = 39     | Remainder: 6    // 62
  39 ÷ 8 = 4      | Remainder: 7    // 762
   4 ÷ 8 = 0      | Remainder: 4    // 4762
```

Octal

- Extremely important for Unix file permission setting;
  - digits used: `0` to `7`, (`0 1 2 3 4 5 6 7`)
- To convert binary to octal, group the binary number in sets of 3, from right to left, then convert each group to an octal digit (pad left with zeros if necessary)
  - 100111110010_2 from previous:

    100 111 110 010
    4 7 6 2

  - 342391_{10}, convert to binary, then:

    001 010 011 100 101 110 111
    1 2 3 4 5 6 7

    1234567_8

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Hexadecimal

- Base 16
- Digits used: 0 1 2 3 4 5 6 7 8 9 A B C D E F
- $9F2_{16}$:

16^2	16^1	16^0
256	16	1
9	F	2

- Converting from Hexadecimal to Decimal:

$$(9 \text{x} 256) + (F \text{x} 16) + (2 \text{x} 1) = 2546_{10}$$
$$9F2_{16} = 2546_{10}$$

Keep in mind in the above example $F = 15$

Decimal to Hexadecimal

- Use the Quotient Remainder method;
- Divide the number by 16 - if there is a remainder, insert remainder, (starting from the right):

```
2546 ÷ 16 = 159    | Remainder: 2    // 2
 159 ÷ 16 = 9      | Remainder: F    // F2
   9 ÷ 16 = 0      | Remainder: 9    // 9F2
```

Hexadecimal

- Almost always used to display memory in computers
- Digits used:

Hex	Binary	Hex	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Binary to Hex Conversion

- Group the binary number in sets of 4, from right to left;
- Convert each group to an hex digit (pad left with zeros if necessary).

- 229656673502_{10} is $0011010101111000100110101011110011011110_2$
- Converting to Hex:

0011	0101	0111	1000	1001	1010	1011	1100	1101	1110
3	5	7	8	9	A	B	C	D	E

Your Turn

- Convert these base 10 numbers to Octal:
  1. 23; and
  2. 56.
- Check your work, convert them back to base 10 numbers.
- Convert these base 10 numbers to Hexadcimal:
  1. 23; and
  2. 56.
- Check your work, convert them back to base 10 numbers.

Signed Integers

Positive and negative in memory

- The most significant bit determines the sign.
  - `0` for positive and `1` for negative
- The weight of each remaining bit is a power of two.
- For negative the weight is the negative of the corresponding power of two.
  - Calculated and stored using two's complement
- Must always know how many bits the number is stored in.
  - 8, 16, 32 or 64.

2’s Compliment

- Calculate two’s complement by, inverting the bits using the bitwise NOT operation;
- Add 1 to the resulting value.
- Decimal 85_{10}:

```
8 bit
Pos Binary:     01010101
 Flip Bits:     10101010
                      +1
Neg Binary:     10101011

16 bit
Pos Binary:     0000000001010101
Neg Binary:     1111111110101011
```

2's Compliment (2)

- Decimal 80_{10}:

```
8 bit
Pos Binary:     01010000
 Flip Bits:     10101111
                      +1
Neg Binary:     10110000
```

Real Numbers

Real Numbers

- Positive or negative value consisting of a whole number plus a fractional part (expressed in floating point, or scientific notation);
- The `float` and `double` data types, are an abstraction of the real numbers that exist in the mathematical world;
- The range and accuracy of real numbers are limited in any computing system;
- Why? How would you store $\frac{1}{3}$ or $\sqrt{2}$?

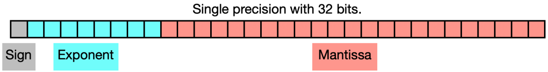
Range and Accuracy of Real Numbers

- Determined by number of bits and the split up of the **mantissa** and **exponent**
- There has to be a limit on the range, by definition, an infinite number of bits needed to represent infinity (∞)
- Accuracy is therefore limited
  - The number of significant digits is limited
    - There are an infinite number of real values between any two points on the number line
  - Irrational numbers
  - Recurring decimals
  - IEEE 754 form (binary conversion)

IEEE 754 (Floating Point) Numbers

- Comprises of sign, exponent and mantissa
  - Mantissa A.K.A Significand

- Single precision - binary32
  - Sign bit: Most Significant bit; 0 pos, 1 neg
  - Exponent width: 8 bits biased to 127
  - Mantissa (significand) precision: 24 bits (23 explicitly stored)



Real Conversion

- To convert a base `10` real number into an IEEE 754 binary32 format use the following outline:

  (**NB** refer to the IEEE 754 standard for strict conversion including rounding behaviour)

- A real number with an integer and a fraction part 12.375
  - Determine the sign bit
  - Positive, hence: `0`
- Convert the integer part into binary
  - 12 = 1100
- Convert the fraction part into binary
  - .375 = .011 (**RELAX**: explained in the following slides)

Fraction Conversion

- Multiply the fraction by 2
- If value >= 1:
  - write a 1 to the mantissa, then
  - subtract 1 from the value,
- Otherwise:
  - write 0 to the mantissa.

- Repeat the preceding steps for the appropriate number significant bits:
  - stop if the value is 0 (after subtracting 1) **or**
  - when the required number of significant bits is reached.

Fraction Conversion (2)

- 0.375

Calc	Result	Manitssa
2 x 0.375	0.75	0
2 x 0.75	1.5	1
1.5 - 1	0.5	--
2 x 0.5	1.0	1
1.0 - 1.0	0	Done

.011

Conversion

- Add the two results
  - 1100.011
- Normalise them
  - Move the decimal point until it is right of the significant bit
  - 1.100011
  - Moved 3 places, used for the base 2 exponent;
  - 1.100011 x 2^3
- Exponent based on precision (127 for single bit precision):
  - $127 + 3 = 130$
  - Convert to binary 10000010
- Grab the mantissa
  - 100011 (Integer part (always 1) is not stored)
- Final encoding is (exact in this case):

Single precision with 32 bits.

Sign	Exponent	Mantissa
0	1 0 0 0 0 0 1 0	1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0