How to THINK like a Programmer

Problem Solving for the Bewildered

paul vickers

Problem Solving 1

starting to think like a programmer

John Brosnan,
Computing Dept, ITT

Aim of Lesson

- This lesson presents an overview of problem-solving:
  - The problem with programming
  - Programs & algorithms
  - Thinking like a programmer
  - Abstraction
  - Pseudocode

The Problem with Programming

- Too often, learning to program is pure bewilderment!

- Blame is placed on a ‘difficult’ programming language
- The trick is to **separate coding from problem-solving**

Why program?

- We use computers to do so many things, including
  - Calculate answers to complex mathematical equations
  - Manage the download of and payment for digital music
  - Schedule classes in the institute
  - Store and edit documents/books/assignments
  - Edit home videos
  - Make music
  - Browse the world wide web

- **Telling the computer what to do and how to do it is called "programming"**

Programs

- **The set of instructions written in a particular programming language that computers can follow are called “programs”**
  - An ATM has a program to tell it how to dispense cash
  - A word processor is a program that tells the computer how to store documents and allow you to edit them
  - The computer (microchip) in an iPod has a program to give you access to your music and video files
  - Your mobile phone has programs to enable it communicate with the cellular network and place & receive phone calls, SMS messages, voicemail etc

Programming

- Before the computer can do what we want we must tell it **how** to perform its task
  - We do this by **first** solving the problem of **deciding how the task can be carried out**
  - Then **expressing that solution in a form that can be turned into something the computer can interpret**.
- The first stage (deciding how the task can be carried out) is the **really important part** as without doing this well we cannot progress to feeding the information into the computer
- Many beginners have a sense of awe, as if a program is some mystical artefact that only those initiated into the dark arts of computer science can produce!

Everyday programming

- A lot of everyday activities use some of the skills needed to program a computer:
  - Cooking a meal for guests: you need to know how many guests so that you can work out what quantities of ingredients are needed; you need a method for making sure everything is cooked and ready at the right time
  - Driving somewhere new: you need to plan a route, estimate the duration of the journey, decide whether the car needs more fuel
  - Getting dressed: where are you going today? What clothes are appropriate? – you don't wear the same clothes to graduation as you would to a beach party; what order do you put your clothes on?
- In all the cases above, **decisions** must be made and things must happen in a certain **order**, typical of many programs

Programs as recipes

- A computer program is like a recipe. A good recipe tells you:
  - what ingredients (and their quantities) are needed
  - how to prepare the ingredients
  - in what order the ingredients must be added
  - what temperature the oven should be set to, if roasting
  - how long to cook everything
- A recipe has a clear method. **An ordered series of steps that must be followed exactly to get the right result**. Recipes are **repeatable**: do it the same each time, get the same results
- A program tells a computer what steps to carry out and in what order, in order to get the correct result

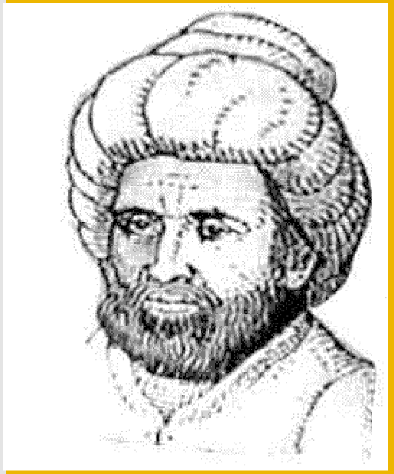
Thinking like a programmer

- To be a programmer you need:
  1. To understand how to examine and analyse problem statements
  2. To understand the component parts of a problem
  3. To understand what is required (what the outcome should be)
  4. To solve the problem
  5. To check the solution for mistakes
  6. To write down your solution in a way that it is clear and easily followed (like a good recipe)
- Only after doing 1–6 above should you consider **translating your solution into programming language code** (Java for us)
- This section of the module deals with steps 1–6

Algorithms

- In computing, an **algorithm is a set of steps designed to solve a particular programming problem**

- On the right is a picture of **Al-Khwarizmi**, a 9th century mathematician, from whom the word “algorithm” derives.



- This section of the module is about learning how to understand various problems and create algorithms that are solutions to those problems.

Key idea: Abstraction

- **Abstraction is vital in programming** - **it's a way of referring to something without revealing all the details – a way of coping with the infinite detail of reality** e.g.

- 'I drove my **car**' this morning:
  - 'car' is an *abstraction* for the particular car I own – no mention of type of car, colour, registration number etc.

- A **map** is a high-level abstraction of a piece of land:
  - Roads, rivers, churches, monuments are shown
  - Probably houses are not shown (unless its highly zoomed)
  - The animals and insect life definitely are not shown

Abstraction

- These are **abstract ways of managing an otherwise unmanageable amount of information**

Abstraction in programming

- **Programmers use abstraction to simplify or hide detail** - often we **defer** dealing with the detail until later, if at all

- In our examples we often start off with fairly general abstractions and **refine** them as we head closer and closer to our final solution

Pseudocode

- In this module we will use a special **notation** called **pseudocode** for writing down solutions to problems

Pseudocode is a form of **structured English**:

- It has features that resemble real programming language code
- But it retains sufficient natural language to allow solutions to be expressed without needing to understand the precise details of a programming language

- We begin with a fairly loose (highly abstract) first draft of pseudocode solution to a problem and progressively refine it as we go on
- This allows us to progress seamlessly from acquiring general problem-solving skills to using quite precise language which is very close to real code.