

# Computers, Programs, Programming Languages

## COMP1100 — Introduction to Programming and Algorithms

Clem Baker-Finch  
Australian National University  
Semester 1, 2006

## What is a computer?

**1940s:** A human being — job description.

**1950s:** John von Neumann — stored program computer.

**Not a calculator:** Before von Neumann there were *calculators* which had a store and a processor. The *instructions* to calculators are externally controlled (by a human).

## Reminders from last lecture:

- Register in prac groups at <http://cs.anu.edu.au/streams/>.
- Make sure you have a copy of the Student Computer Environment User Guide.
- Work through the Student Computer Environment familiarisation exercises **before your first prac class** (week 2).
- Read chapter 1 of the textbook.

## Von Neumann's brilliant idea:

- Store + processor
- Data **and instructions** in store — the instructions are the **program**.
- Two aspects to the processor:
  - Fetch instructions, decode and execute the instructions
  - Do the calculations to modify the data in the store

By loading the instructions into the computer, it can complete the calculation **independently**.

In contrast, calculators require separate control and supervision.

## Binary representation

In 1697, Leibniz discovered the binary numeral system and binary computation.

(Binary: base-2 number representation. Digits 0 and 1 only.)

Convenient for electronic devices:  
charge versus no charge; current versus no current.

*All* data is represented using binary digits (bits). E.g. in the ASCII encoding, character 'A' has the same representation as integer 65.

## Modern programming languages

Fortran and LISP are still widely used.

Programs are now much, much more complex —  
languages are now (supposedly) designed to help manage that complexity.

There are hundred of different programming languages – some general purpose, most oriented towards a particular problem domain.

## What is programming?

The *human activity* of designing and constructing the instructions to make a computer achieve some particular task.

## What is a programming language?

Writing programs as sequences of machine instructions in binary notation is obviously very inconvenient for human programmers.

A programming language is a notation for expressing a program.

**Fortran:** The first programming language (late 1950s). Based closely on the operations of a particular machine (IBM 704).

**LISP:** The second programming language (late 1950s). Based on the lambda calculus — programs consist of evaluations of expressions and applications of functions.

## Kinds of programming languages

**Imperative:** A sequence of commands —  
Step-by-step description of intended changes to the computer store.

**Declarative:** Focus on *what* is to be computed rather than the details of *how* this is achieved.

**Functional:** Subset of the declarative languages —  
Computation is expressed as *functions* from input to output values.  
**Haskell** is a functional language.

**Object-oriented:** Usually imperative, but not necessarily —  
Structure programs around the idea of *objects* and messages.  
**Java** is an imperative object-oriented language.

## Translation

Computers can only execute its own set of instructions.

No matter what language we use to write our programs, we eventually want to run it on a computer. So our programs must be *translated* to the instructions of the machine.

We use a **program** called a *compiler* to do the translation.

The input to the compiler is a program written in a high-level language. The output is the 'same' program as machine instructions.

For Haskell we will be using the **Glasgow Haskell Compiler, GHC** .

## Why Haskell?

- It is **easy to get started** writing Haskell programs.  
The simplest Java program requires you to know about I/O, libraries and many syntactic details.
- GHC has an interactive interface that works more like a calculator. We don't need to write whole programs.
- It is **easy to understand** how Haskell programs work — they evaluate expressions. To understand Java programs, first you need to understand the underlying *machine model*.

## A Haskell program

HelloYou.hs is a little Haskell program. Don't worry about trying to understand it for now.

We can use GHC to *translate* HelloYou.hs to machine instructions. Type:

```
ghc HelloYou.hs
```

This will create some new files, including `a.out` which is the machine language version of HelloYou.hs.

**Run** this program by typing `a.out`  
(or possibly `./a.out`).

## Why Haskell? (ctd)

- Problem-oriented **data structures** are easy to define in Haskell.
- **Types** are a fundamental organising principal in programming languages. Haskell's type system is the most advanced and well-designed of *any* programming language.
- The aim is not to become highly skilled Haskell programmers.  
The aim is to learn some **fundamental principles of programming**.