SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. Ridha Khedri

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

- Computer technology changes with a frightening speed
- HOWEVER, the fundamentals remain remarkably static
  - The architecture of standard computer is hardly changed from the machine the machine which were built half a century ago
  - The mathematics needed to deal with the common problems is also quite old
  - The design principle that we abide by when solving problems are as old as engineering (with some specificities to software systems)
- What doesn't change in the programming activity?

- Is a programmer a translator from a language to another?
- Is a programmer someone who knows only about programming language(s)?
- Is a programmer a system builder?
  - Defines the problem to be solved
  - Develop the system and evaluate alternatives
  - Asserts that the provided system is correct w.r.t. what is required
  - Ensures that the system fits well its intended environment
  - Ensures that the system is **sustainable**

When we build, let us think that we build forever. Let it not be for present delight nor for present use alone. Let it be such work as our descendants will thank us for; and let us think, as we lay stone on stone, that a time is to come when those stones will be held sacred because our hands have touched them, and that men will say, as they look upon the labor and wrought substance of them, "see! This our father did for us." [John Ruskin (8 February 1819 – 20 January 1900)]

(1) The programming language

- What is a language?
- What are its elements?
  - Its syntax: The set of rules that define the combinations of symbols that are considered to be correctly structured programs in that language
  - Its semantics: The rigorous mathematical meaning of the programming language

What is a language?

- A set of strings on an alphabet Σ (i.e., a subset of Σ^*)
- Can we be more precise?
  - It is generated by a grammar G (we often denote it by $L(G)$)
- What is a grammar?

An example of a grammar

<stmt> ::= <if-stmt> | <while-stmt> | <begin-stmt> | <assg-stmt>
<if-stmt> ::= **if** <bool-expr> **then** <stmt> **else** <stmt>
<while-stmt> ::= **while** <bool-expr> **do** <stmt>
<begin-stmt> ::= **begin** <stmt-list> **end**
<stmt-list> ::= <stmt> | <stmt> ; <stmt-list>
<assg-stmt> ::= <var> := <arith-expr>
<bool-expr> ::= <arith-expr><compare-op><arith-expr>
<compare-op> ::= $<$ | $>$ | $\leq$ | $\geq$ | $=$ | $\neq$
<arith-expr> ::= <var> | <const> | (<arith-expr><arith-op><arith-expr>)
<arith-op> ::= $+$ | $-$ | $*$ | $/$
<const> ::= **0** | **1** | 2 | **3** | **4** | 5 | 6 | 7 | 8 | 9
<var> ::= a | b | c | $\cdots$ | x | y | z

What is needed to be a good programmer?

What a programmer needs to know?

- The objects $\langle xxx \rangle$ are called nonterminal symbols
- Each nonterminal symbol generates a set of strings over a finite alphabet Σ in a systematic way
- the nonterminal $\langle$arith-expr$\rangle$ in

$$\langle \text{assg-stmt} \rangle ::= \langle \text{var} \rangle := \langle \text{arith-expr} \rangle$$

  generates the set of syntactically correct arithmetic expressions in this language

- The strings corresponding to the nonterminal $\langle xxx \rangle$ are generated using rules with $\langle xxx \rangle$ on the left-hand side

- The alternatives on the righthand side, separated by vertical bars $|$, describe different ways strings corresponding to $\langle xxx \rangle$ can be generated
- These alternatives may involve other nonterminals $\langle yyy \rangle$, which must be further eliminated by applying rules with $\langle yyy \rangle$ on the left-hand side
- while $x \leq y$ do begin $x := (x + 1);\ y := y - 1$ end is generated by the nonterminal $\langle stmt \rangle$

- We obtain the while statement from ⟨stmt⟩ through a sequence of expressions called sentential forms
- Each sentential form is derived from the previous by an application of one of the rules

⟨stmt⟩
⟨while-stmt⟩
while ⟨bool-expr⟩ do ⟨stmt⟩
while ⟨arith-expr⟩ ⟨compare-op⟩ ⟨arith-expr⟩ do ⟨stmt⟩
while ⟨var⟩ ⟨compare-op⟩ ⟨arith-expr⟩ do ⟨stmt⟩
while ⟨var⟩ $\leq$ ⟨arith-expr⟩ do ⟨stmt⟩
while ⟨var⟩ $\leq$ ⟨var⟩ do ⟨stmt⟩
while $x \leq$ ⟨var⟩ do ⟨stmt⟩
while $x \leq y$ do ⟨stmt⟩
while $x \leq y$ do ⟨begin-stmt⟩
...

- Applying different rules will yield different results

$$\text{begin if } z = (x + 3) \text{ then } y := z \text{ else } y := x \text{ end}$$

- The set of all strings not containing any nonterminals generated by the grammar is called the language generated by the grammar
- In general, this set of strings may be infinite, even if the set of rules is finite
- There may also be several different derivations of the same string
- A grammar is said to be unambiguous if a string cannot have more than one derivation

Definition

A context-free grammar (CFG) is a quadruple $G = (N, \Sigma, P, S)$, where

- N is a finite set (the nonterminal symbols),
- Σ is a finite set (the terminal symbols) disjoint from N,
- P is a finite subset of $N \times (N \cup \Sigma)^*$ (the productions),
- $S \in N$ (the start symbol).

- The grammar gives the syntax of the language
- The semantics of the language is a more complex issue
  - It requires quite advanced mathematical background
  - At this level, we deal with it in an informal way

What else a programmer needs to know?

(2) A wealth of commonly known algorithms and data structures

- Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output

- Examples of known algorithms: Sorting, search, graph (e.g., spanning trees, shortest path, maximum flow, coloring), etc.

- We can view algorithms as a tool for solving a well-specified computational problem
- The statement of the problem specifies in general terms the desired input/output relationship
  - The precondition: specifies the set of acceptable input
  - The postcondition: specifies the set of acceptable output
- It is very important to know the strengths and limitations (e.g., complexity) of several of them

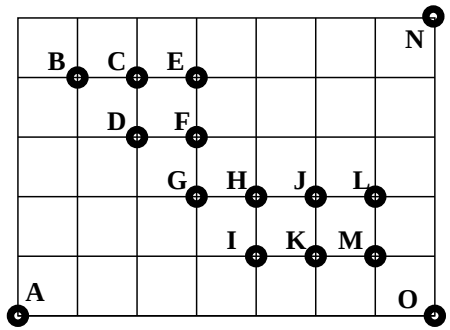
- A data structure is a way to store and organize data in order to facilitate access and modifictions
- Examples of data structures: Arrays, stacks and queues, hash tables, graphs, heaps, etc.
- No single data structure works well for all purposes
- Therefore, it is very important to know the strengths and limitations of several of them

Where you get this knowlege?

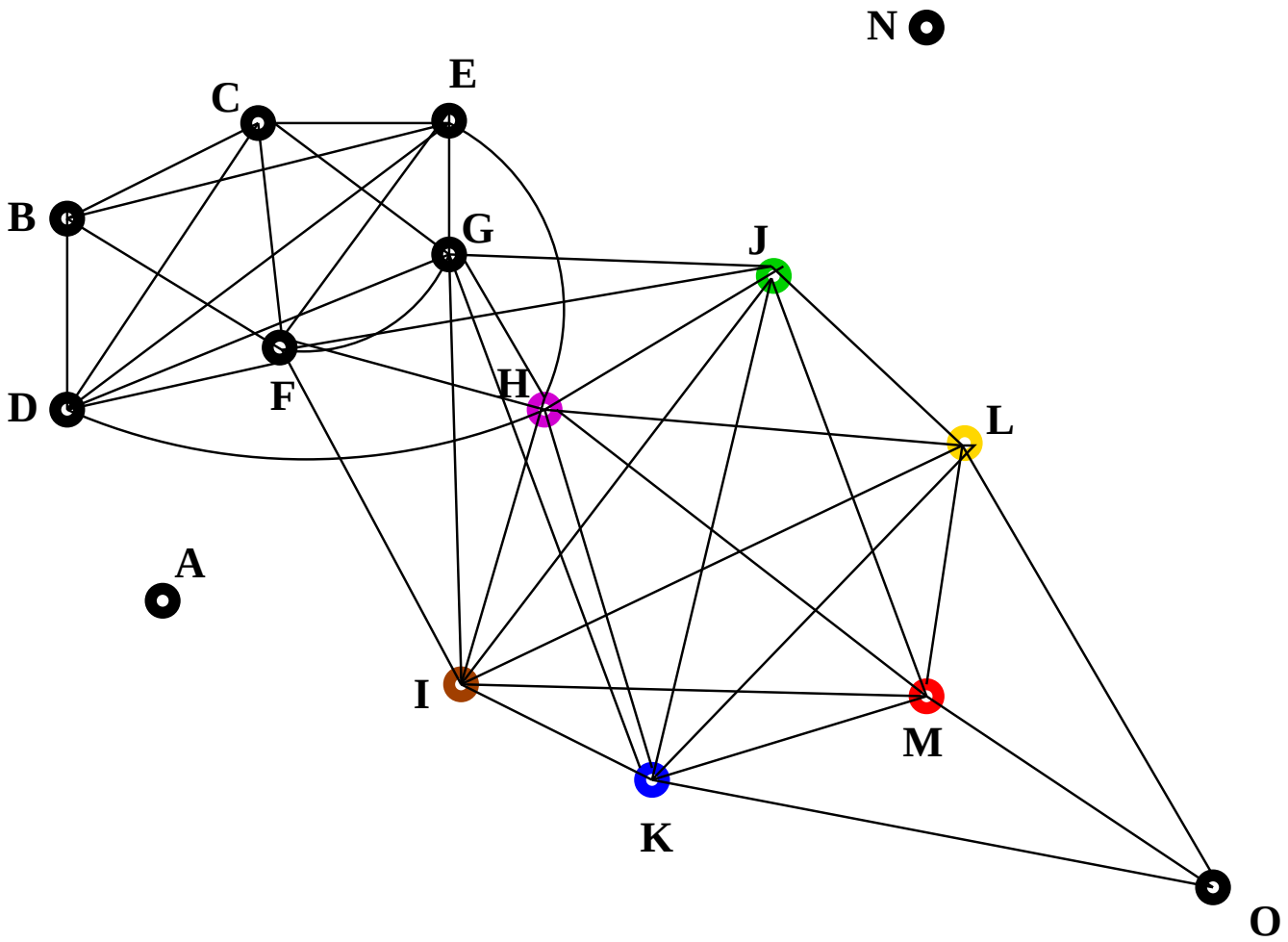
- Some of it in this course
- Algorithms course
- Maths courses (as we see in the next example)

Programming Overview

Television channels are assigned to broadcasting stations by a governmental agency. Obviously, two stations in geographic proximity must get different channels, to avoid reception interference. Suppose that the rule has been adopted that stations within 140 miles of each other (as the crow flies) must have different channels. The grid shows the locations of 15 hypothetical stations. Each square is 50 miles on a side. How many channels are required, and how can they be assigned to comply with the rule?



Programming Overview



(3) Software Design Skills

- During the last half centery, the world witnessed growth in the scale, complexity, and space distribution of software systems
- When a system is complex in its functionality, its aspects (qualities), and its environment, an obvious need to reduce that complexity emerges
- A top-down strategy in software construction to divide and conquer the complexities faced by the constructors
- Roughly speaking, the decomposition of a system/artifact into "manageable" parts is one of aims of design

What is Design?

- One cannot define design as would define a mathematical activity such as “rewriting“, “derivation”, etc.
- The nature of design is not easy to grasp in a clear definition
- Looking for a definition of design may not help you grasp what it is
- Design can be seen in any human artefact and in nature
- The single word “design” encompasses an awful lot of objectives and subjective aspects

- They can be
  - aesthetic (subjective)
  - functional (Objective)
  - many other aspects of an object or a process (Objective or subjective), which usually requires considerable
    - research,
    - thought,
    - modeling,
    - interactive adjustment,
    - and re-design

- Depending on the designed entity, either the objective or the subjective aspects might takeover
- When the subjective aspects take over, design is then viewed as a more rigorous form of art, or art with a clearly defined purpose
- When the objective aspects take over, design is then viewed as a simple mathematical transformation of the mathematical entities expressing the functional requirements into a design which is a collection of mathematical entities with their mathematical connectors
- So, design involves both mathematics and art as transformation means (and may be other skills)

What is in design more than the translation of an idea to an artefact or process?

- Design can be seen as an activity of reconciliation between conflicting needs or constraints put by the environment
- Design is a creative process based around the "building up" of ideas
  - Avoiding negative judgments of contributors when designing eliminates the fear of failure and encourages maximum input and participation
  - Encourage outside the box thinking in this process since this can often lead to creative solutions

- A bad software design may result in
  - a poor quality software product which does not meet user requirement,
  - A system that is not adaptive to any further requirement changes
  - A system that is not reusable
  - A system that has unpredictable behavior
  - A system that has poor performance
  - Software production itself may be very ineffective in terms of time and cost

(4) Know the environment for your programs (CPU, Memory, etc..)

- The various components of a computer hardware (e.g., CPU, memory, mass storage, and network interfaces) are the parts of the environment for your programmes
- As CPU cores become both faster and more numerous, the limiting factor for most programs is now, and will be for some time, memory access
- Hardware designers have come up with ever more sophisticated memory handling and acceleration techniques
- However, these techniques cannot work optimally without some help from the programmer
- Therefore, the programmer ought to have some knowledge of these techniques

- We will be looking at these 4 sets of skills/knowledge while we are studying Java programming language

- We will be paying attention to the following:
  - Good programming practices (We will use a programming style guide)
  - Common programming errors
  - Testing and debugging tips
  - Performance tips
  - portability tips
  - Software engineering observations (design issues, sustainability issues, etc.)
  - Look-and-feel observations

Java provides the engineer and scientist with an

- efficient and easy-to-use tool for problem solving in today's web-based environment
- A small and efficient core language
- Full graphics, event driven programming and multimedia
- Genuine platform independence and the ability to transmit and run programs over the internet
- Facilities for parallel programming and for communicating on the network in a variety of ways