Programming Language Translation Lecture 1(cont.)

Karen Bradshaw

Covers Chapter 1 of the notes

Some definitions

- Taken from Chapter 1 of the notes
- Please ensure you understand all of these as they are the foundation of what follows

Systems programs

Those programs that allow for the easier development of other programs and systems

- Compilers, assemblers
- Interpreters, emulators
- Linkers
- Editors
- File managers
- Operating systems

Systems programs (2)

- ***Compilers*** and ***assemblers*** are systems programs that translate a program written in one language (usually problem-oriented) into another language (usually machine oriented).

Computer languages

- Notations for describing how the solution to a problem posed at one level and in one domain may ultimately be effected by a processor or processors working at another level and in another, highly rigorous domain

- What these notations have in common is that they attempt to bridge a *semantic gap*

Computer languages (2)

- **High-level languages**
  - Notation may be quite close to being an abstract mathematical notation
  - Classified as:
    - Procedural (Fortran, Ada, C, Pascal or Modula-2)
    - Object-oriented (C++, Java or C#)
    - Functional (Lisp, Scheme, ML or Haskell)
    - Logic (Prolog)

- **Low-level languages**
  - Notation tends to be little more than a list of atomic machine level instructions
    - Assembly language

Computer language pitfalls

Language is used by humans to

- Formulate ideas
- Develop ideas
- Express and explain ideas

- So one's understanding, knowledge and experience of a language determines ***how*** as well as ***what*** one can think (or how and what one can ***program***, in the case of computer languages).

- It is easy to confuse the abstract notion of a "computer language" with the notion of an ***implementation*** of that language, since one cannot make any real use of a language if the implementation does not exist.

Computer language design

- Two ways to devise a computer system or a computer language
- As Tony Hoare put it:
  "There are basically two routes to follow. One is to strive for a design that is so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies."
- Historically there have been cycles – simple gives way to complex, which gives way again to simple, and so on.
- Hoare again:
  "Algol 60 was not only a great improvement on its predecessors, but also on nearly all of its successors."
- One is reminded of the maxim attributed to Albert Einstein:
  "Make it as simple as you can, but no simpler."

Desirable features: programmer

- ***Readability***
  - Source code resembling natural language
- ***Familiarity***
  - Decimal arithmetic, expressions resembling mathematics
- ***Portability***
  - Source code should be machine independent
- ***Generality***
  - Should allow a wide variety of applications to be programmed
- ***Brevity***
  - Should allow compact source code (powerful statements)
- ***Easily learned***
  - Successful languages are often based on other familiar languages
- ***Error checking***
  - Provide features that encourage programs to help debug themselves (range, subscript and null pointer checking)

Desirable features: implementer

- ***Orthogonality***
  - There should not be a confusing number of ways of doing the same thing - language features should not interact badly or be arbitrarily restricted
- ***Clearly defined syntax and semantics***
  - There must be no room for doubt, and no ambiguities
- ***Modularity***
  - Support for the development of large systems in terms of smaller, easily verified subsystems
- ***Efficient***
  - Language features should permit the generation of efficient object code
- ***Easily translated***
  - Preferably only one or two "passes" should be needed
  - Preferably handled by deterministic parsing without backtracking

Syntax and semantics

For natural languages:

- ***Syntax is***
  - *"that part of grammar that deals with the construction of sentences and the correct arrangement of the words therein; the rules governing sentence construction"*
- ***Semantics is***
  - *"the study of meaning; pertinent to the meaning of words"*

In programming languages/notations, "words" and "sentences" become "statements" and "programs", respectively

Thus, for programming language translation:

- ***Static Semantics***
  - The aspects of meaning that can be deduced, and checked, at *compile-time* (that is, before the program is actually executed)
- ***Dynamic Semantics***
  - The aspects of meaning that are only pertinent, and can only be checked, at *run-time*, when an attempt is made to execute the program