CIT3136 Programming Language Implementation

- **Lecturer** (Professor) Rodney Topor
  rwt@cit.gu.edu.au, TB 2.20
- Lectures: Mondays @ 11.30, HMN 0.10
- **Tutor** To be determined
- Tutorial/Laboratories: Thursdays @ 4.30-6.30, TEN 2.16
- **One line summary** Basic principles and practice of design and implementation of interpreters and compilers for programming languages.
- **Assessment** 60% programming assignment, 40% examination
- This is a computer science subject!

**Motivation**

- Learn history of programming language design and implementation.
- Learn design principles of programming languages.
  (Every user interface is a programming language)
- Learn effective and efficient use of programming languages.
- Understand language-machine interactions.
  (Interface implementation often involves abstract machine design.)
- Case study in large-scale software engineering.
  (Implementations are large, complex programs.)
- Extend software engineering and programming skills.
- Extend mathematical reasoning skills.
- Enjoy challenge of learning new concepts and completing challenging assignment.
References

- S.N. Kamin, Programming Languages: An Interpreter-Based Approach, Addison-Wesley, 1990.
- Many, many more.

Programming languages

- VisiCalc (1980?), Troff, Visual Basic/C/C++, \LaTeX, HTML, XML, Word/Excel macros, ...
- sh, csh, bash, DOS (ugh), MacOS, Win9x, ...
Interpreters and compilers

- **Compilers**
  - Translation of program in source language to target machine language.
  - More precisely, translation of module in source language to linkable object file in target machine language.

- **Interpreters**
  - Direct execution of source program.
  - Compilation into abstract/virtual target machine language, followed by interpretation/emulation of target program.

A simple expression interpreter
A simple statement translator

Model compiler structure

Components

- Structural analysis: scanning (lexical analysis), parsing (syntactic analysis)
- Semantic/contextual analysis: identification, type analysis
- Code generation: instruction selection, register allocation, ...
- Error detection, recovery and reporting
- Optimisation

Program representations

- Source program
- Token stream
- Abstract syntax tree (source-oriented)
- Attributed AST (or target-oriented tree)
- Linkable object file (symbolic target machine code)
- Executable file (absolute target machine code)

Run-time organisation

- Data representation
- Procedure frame management
- Dynamic storage management
Exercise

Write a lexical analyser for a subset of C.

Input: Stream of characters.

Output: Stream of token structures: (token-type, token-value)

(The type of the token-value depends on the token-type.)