How to Write a Compiler

ASU Textbook Chapter 11

Tsan-sheng Hsu

tshsu@iis.sinica.edu.tw

http://www.iis.sinica.edu.tw/~tshsu
Planning

The best and complete way to learn to write a compiler is to
- take a compiler course for the “theory”;
- read the code of a compiler;
- write a compiler by yourself.

The planning stage:
- Source language issues:
  - The size of the language.
  - Will the language evolve?
- Target language issues:
  - Instruction set.
  - Registers.
  - Fancy instructions.
- Performance criteria:
  - Changes come from the hardware development.
  - Portability.
  - Error correction: for both expert and novice users.
  - Optimization.
Developing

- Find an existing language and adapt it for your needs.
- If you read some UNIX C (respectively PASCAL) compiler, they are written in C (respectively, PASCAL).

- This is called **bootstrapping**.
- How can this be possible and how was the first compiler compiled?
- Usual strategy:
  - Find an existing compiler (could be an assembly language).
  - Write a simple compiler for a fairly restricted subset of language.
  - For example in PASCAL, does not allow ARRAY, RECORD, POINTER.
  - Call this a **restricted language**.
  - Write in the restricted language a compiler, that handles advanced features.
  - Another example: C and C++.
Developing environment

- Developing environment:
  - Use UNIX “make” to manage a project.
  - Use lexical analyzer (LEX) and compiler-compiler (YACC) to simplify your task.
  - Use “profile” to determine the bottleneck of implementation.

- Testing and maintenance:
  - Must generate correct code.
  - Regression tests:
    - Maintain a series of tests of which must be passed after.
    - Re-pass the suite of tests once a revision is done to the compiler.
  - Documentation.

- A crucial element in being able to maintain a compiler is good programming style and documentation.