Introduction to Compilers

Adapted from:
Lectures by Profs. Alex Aiken and George Necula
(University of California at Berkeley)
Lectures by Prof. Saman Amarasinghe
(Massachusetts Institute of Technology)

How are Languages Implemented?
• Two major strategies:
  ➢ Interpreters
    • LISP, bash, java, ...
  ➢ Compilers
    • gcc, javac, ...
• Interpreters run programs “as is” (Little preprocessing)
  ➢ Carry-out the meaning of a program.
• Compilers do extensive preprocessing
  ➢ Transform a program in a (higher-level) language into an efficient program in a (lower-level) language, preserving the meaning.

Standard Text Books
• Dragon Book
  ➢ Alfred Aho, Ravi Sethi, and Jeffrey Ullman
• Tiger Book
  ➢ Andrew Appel
• Whale Book
  ➢ Steve Muchnick

Anatomy of a Compiler
Structure of the Compiler vis a vis the two Courses

1. Lexical Analysis
2. Parsing
3. Semantic Analysis
4. Optimization
5. Code Generation

- CS780 deals with the theory and the practice of (1), (2) and if time-permitting, (3).
- CS781 deals with the theory and the practice of (3), (4), and (5).

Running the program

- Preprocess and Compile:
  ```
  gcc -c a.c b.c; gcc -o myc.o c.c
  ```
  Source files to object files
- Link:
  ```
  gcc a.o b.o myc.o
  ```
  Object files to an executable file (a.out)
- Symbol Resolution (and Relocation)
- Shared Libraries
  - Static Linking: (e.g., I/O)
  - Dynamic Linking: (e.g., DLLs, lib.so, Java, …)
- Load and Execute: a.out
  Move binaries from disk to memory and run
  Relocation (and Dynamic linking)

Lexical Analysis

- Recognize words.
- Note the
  - Capital “T” (start of sentence symbol)
  - Blank “ ” (word separator)
  - Period “.” (end of sentence symbol)
- Lexical analyzer divides program text into “tokens”
  ```
  if x == y then z = 1; else z = 2;
  ```
- Units:
  ```
  if, x, ==, y, then, z, =, 1, ;, else, z, =, 2, ;
  ```

Parsing: Diagramming a Sentence

- This line is a longer sentence.
- Subject
- Article
- Adjective
- Noun
- Verb
- Adjective
- Noun
- Sentence
Parsing Programs

A parser recognizes higher-level structure from a token sequence.

• Consider:
  
  \[ \text{if } x == y \text{ then } z = 1; \text{ else } z = 2; \]

• Diagrammed:

Semantic Analysis in English

• Understanding meaning and performing consistency checks.

• Example:
  
  Jack said Jerry left his assignment at home.
  
  What does “his” refer to? Jack or Jerry?

• Even worse:
  
  Jack said Jack left his assignment at home?
  
  How many Jacks are there?
  
  Which one left the assignment?

Semantic Analysis in Programming

• Programming languages define strict rules to avoid such ambiguities
  
  ➢ Scope Rules

• This C++ code prints “4”; the inner definition is used

More Semantic Analysis

• Compilers perform many semantic checks besides variable bindings.

• Example:
  
  Jack left his homework at home.
  
  Jack left her homework at home.

  ➢ A “type mismatch” between her and Jack; we know they are different people.

  ➢ Presumably, Jack is male.

• Example:
  
  \[ \text{int } i, j; \text{ float } x; \]
  
  \[ j = (\text{int}) \ (\ (i + j) \ * \ x); \]
Optimization

- No strong counterpart in English, but akin to editing.
- Automatically modify programs so that the equivalent program
  ➢ Runs faster.
  ➢ Uses less memory.
  ➢ In general, conserves some resource.
- Simple Example:
  \[ X = Y + Y \text{ is the same as } X = 2^*Y \]
  (Implemented as arithmetic left-shift operation)

Code Generation

- Produces assembly code (usually).
- Many compilers perform translations between successive intermediate forms.
  ➢ All but the first and the last are ILs internal to the compiler, typically ordered in descending level of abstraction.
    ❍ Highest is the source language.
    ❍ Lowest is the assembly language.
  ➢ Lower levels expose features such as registers, memory layouts, etc hidden by higher-levels.
  ➢ Lower levels obscure high-level meaning.

Other Issues

- Example: How are errors detected and diagnosed? Error recovery?
- Language design has big impact on compiler.
  ➢ Determines what is easy and hard to compile.
    ❍ Lexical Analysis in FORTRAN/PL-I vs the same in Pascal/Java
  ➢ Many trade-offs in language design.
    ❍ Pointers, Multiple Inheritance,… in C++ vs
    ❍ Strong Typing, Garbage Collection,… in Java