

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)

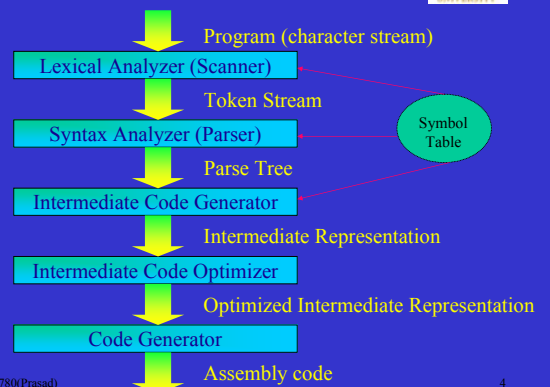
How are Languages Implemented?

- 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



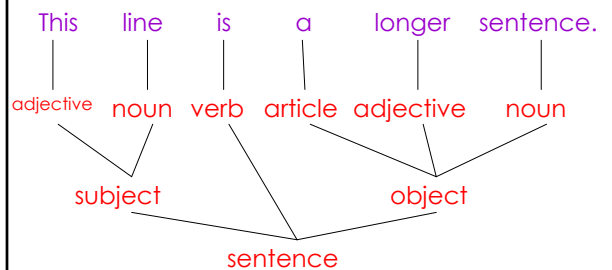
- **Preprocess and Compile :**
`gcc -c a.c b.c; gcc -o myc.o c.c`
 - Source files to object files
- **Link :** `gcc a.o b.o myc.o`
 - Object files to an executable file (a.out)
 - ❑ Symbol Resolution (*and Relocation*)
 - ❑ Shared Libraries
 - ❖ Static Linking : (e.g., l/O)
 - ❖ Dynamic Linking : (e.g., 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.
`This is a sentence.`
- 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



Parsing Programs

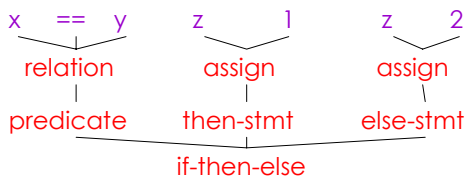


A parser recognizes higher-level structure from a token sequence.

- Consider:

```
if x == y then z = 1; 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

```
{
  int Jack = 3;
  {
    int Jack = 4;
    cout << Jack;
  }
}
```

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:

```
int i, j; float x;    j = (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:
 $X = Y + Y$ 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.

Unoptimized Code

```
lida $30, -32($30)
strq $26, 0($30)
strq $15, 8($30)
bim $30, $30, $15
bim $16, $30, $15
stl $1, 16($15)
lida $4, 16($15)
sta $4, 24($15)
ldi $5, 24($15)
bim $5, $5, $2
s4addq $2, 0, $3
ldi $4, 16($15)
mull $4, $3, $2
ldi $3, 16($15)
addq $3, 1, $4
mull $2, $4, $2
addq $3, 1, $4
addq $3, 1, $4
mull $2, $4, $2
stl $2, 20($15)
ldi $0, 20($15)
br $31, $20
033: bim $15, $15, $30
ldq $26, 0($30)
ldq $15, 8($30)
addq $30, $30, $30
ret $31, ($26), 1
```

Source Code

```
int expr(int n) {
    int d;
    d = 4 * n * n
        * (n + 1)
        * (n + 1);
    return d;
}
```

Optimized Code

```
s4addq $16, 0, $0
mull $16, $0, $0
addq $16, 1, $16
mull $0, $16, $0
mull $0, $16, $0
ret $31, ($26), 1
```

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