Introduction to Compiler Design

Lecture 1
Chapters 1 and 2

Robb T. Koether

Hampden-Sydney College

Wed, Jan 14, 2015

Outline

1. The Stages of Compilation
    - Lexical Analysis
    - Syntactic Analysis
    - Semantic Analysis
    - Intermediate Code Generation
    - Optimization
    - Machine Code Generation

2. Assignment

Outline

1. The Stages of Compilation
   - Lexical Analysis
   - Syntactic Analysis
   - Semantic Analysis
   - Intermediate Code Generation
   - Optimization
   - Machine Code Generation

2. Assignment

The Stages of Compilation

- The stages of compilation
  - Lexical analysis
  - Syntactic analysis.
  - Semantic analysis.
  - Intermediate code generation.
  - Optimization.
  - Machine code generation.

Outline

1. The Stages of Compilation
   - Lexical Analysis
   - Syntactic Analysis
   - Semantic Analysis
   - Intermediate Code Generation
   - Optimization
   - Machine Code Generation
2. Assignment

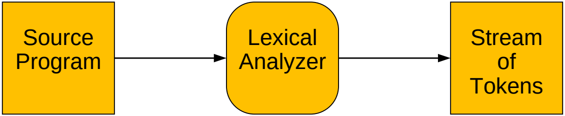
Lexical Analysis

Definition (Token)

A token is a smallest meaningful group symbols.

Definition (Lexical analyzer)

A lexical analyzer, also called a lexer or a scanner, receives a stream of characters from the source program and groups them into tokens.


Source Program

Lexical Analyzer

Stream of Tokens

Example

Example (Lexical Analysis)

- What are the tokens in the following program?

```
int main()
{
    float a = 123.4;
    return 0;
}
```

Tokens

- Each token has a type and a value.
- For example,
  - The variable `count` has type **id** and value “`count`”.
  - The number `123` has type **num** and value “123”.
  - The keyword **`int`** has type **int** and value “int”.
  - The symbol `{` has the type **lbrace** and value “lbrace”.

Example

Example (Lexical Analysis)

- The statement

  ```
  position = initial + rate * 60;
  ```

  would be viewed as

  $$\mathbf{id}_1 \ = \mathbf{id}_2 + \mathbf{id}_3 \ * \ \mathbf{num} \ ;$$

  or

  $$\mathbf{id}_1 \ \mathbf{assign} \ \mathbf{id}_2 \ \mathbf{plus} \ \mathbf{id}_3 \ \mathbf{times} \ \mathbf{num} \ \mathbf{semi}$$

  by the lexer.

Lexical Analysis Tools

- There are tools available to assist in the writing of lexical analyzers.
  - `lex` - produces C source code (UNIX).
  - `flex` - produces C source code (gnu).
  - `JLex` - produces Java source code.
  - `JFlex` - produces Java source code.
- We will use `JFlex`.

Outline

1. The Stages of Compilation
   - Lexical Analysis
   - Syntactic Analysis
   - Semantic Analysis
   - Intermediate Code Generation
   - Optimization
   - Machine Code Generation

2. Assignment

Syntactic Analysis

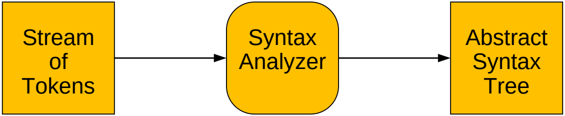
Definition (Syntax analyzer)

A syntax analyzer, also called a parser, receives a stream of tokens from the lexer and groups them into phrases that match specified grammatical patterns.

Syntactic Analysis

Definition (Abstract syntax tree)

The output of the parser is an abstract syntax tree representing the syntactical structure of the tokens.



Stream of Tokens

Syntax Analyzer

Abstract Syntax Tree

Grammatical Patterns

- Grammatical patterns are described by a context-free grammar.
- For example, an assignment statement may be defined as

$$\begin{aligned} \textit{stmt} &\to \textbf{id} \ = \ \textit{expr} \ ; \\ \textit{expr} &\to \textit{expr} \ + \ \textit{expr} \mid \textit{expr} \ * \ \textit{expr} \mid \textbf{id} \mid \textbf{num} \end{aligned}$$

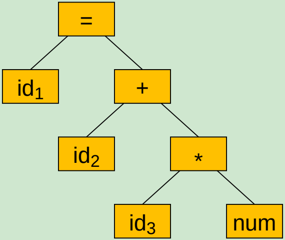
Example

Example (Syntactic Analysis)

- The form

$$\mathbf{id}_1 = \mathbf{id}_2 + \mathbf{id}_3 * \mathbf{num} \; ;$$

may be represented by the following syntax tree.



Syntax Analysis Tools

- There are tools available to assist in the writing of parsers.
  - `yacc` - produces C source code (UNIX).
  - `bison` - produces C source code (gnu).
  - `CUP` - produces Java source code.
- We will use `CUP`.

Outline

1. The Stages of Compilation
   - Lexical Analysis
   - Syntactic Analysis
   - Semantic Analysis
   - Intermediate Code Generation
   - Optimization
   - Machine Code Generation

2. Assignment

Semantic Analysis

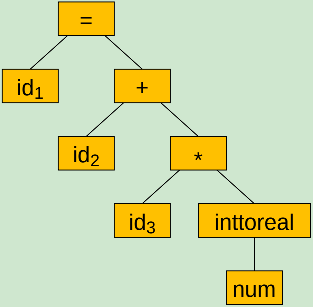
Definition (Semantic analyzer)

A semantic analyzer traverses the abstract syntax tree, checking that each node is appropriate for its context, i.e., it checks for semantic errors. It outputs a refined abstract syntax tree.

Example: Semantic Analysis

Example (Semantic Analysis)

- The previous tree may be refined to



Outline

1. The Stages of Compilation
   - Lexical Analysis
   - Syntactic Analysis
   - Semantic Analysis
   - Intermediate Code Generation
   - Optimization
   - Machine Code Generation

2. Assignment

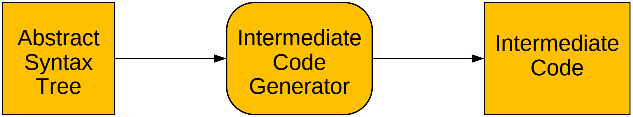
Intermediate Code Generation

Definition (Intermediate code)

Intermediate code is code that represents the semantics of a program, but is machine-independent.

Definition (Intermediate code generator)

An intermediate code generator receives the abstract syntax tree and outputs intermediate code that semantically corresponds to the abstract syntax tree.



Abstract Syntax Tree

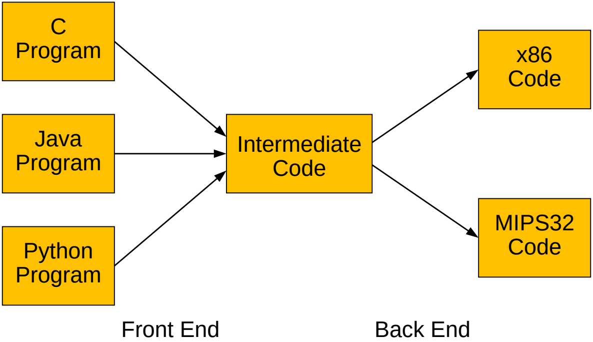
Intermediate Code Generator

Intermediate Code

Intermediate Code

- This stage marks the boundary between the front end and the back end.
- The front end is language-specific and machine-independent.
- The back end is machine-specific and language-independent.

Intermediate Code



Example

Example (Intermediate Code Generation)

- The tree in our example may be expressed in intermediate code as

```
temp1 = inttoreal(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```

Outline

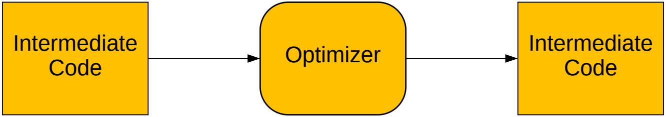
1. The Stages of Compilation
   - Lexical Analysis
   - Syntactic Analysis
   - Semantic Analysis
   - Intermediate Code Generation
   - Optimization
   - Machine Code Generation
2. Assignment

Code Optimizer

Definition (Optimizer)

An optimizer reviews the code, looking for ways to reduce the number of operations and the memory requirements.

- A program may be optimized for speed or for size.
- Typically there is a trade-off between speed and size.



Example

Example (Optimization)

- The intermediate code in this example may be optimized as

```
temp1 = id3 * 60.0
id1 = id2 + temp1
```

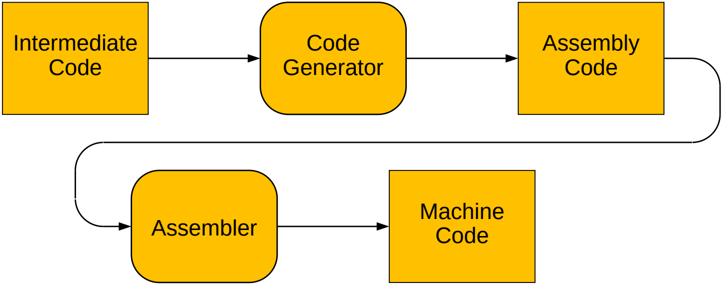
Outline

1. The Stages of Compilation
   - Lexical Analysis
   - Syntactic Analysis
   - Semantic Analysis
   - Intermediate Code Generation
   - Optimization
   - Machine Code Generation
2. Assignment

Machine Code Generation

- The code generator receives the (optimized) intermediate code.
- It produces either
  - Machine code for a specific machine, or
  - Assembly code for a specific machine and assembler.
- If it produces assembly code, then an assembler is used to produce the machine code.

Machine Code Generation



Example: Machine Code Generation

- The intermediate code may be translated into the assembly code

```
movf id3,R2
mulf #60.0,R2
movf id2,R1
addf R2,R1
movf R1,id1
```

Outline

1. The Stages of Compilation
   - Lexical Analysis
   - Syntactic Analysis
   - Semantic Analysis
   - Intermediate Code Generation
   - Optimization
   - Machine Code Generation

2. Assignment

Assignment

Assignment

- Read Chapters 1 and 2.