Concepts Introduced in Chapter 2

- A more detailed overview of the compilation process.
  - Parsing
  - Scanning
  - Semantic Analysis
  - Syntax-Directed Translation
  - Intermediate Code Generation

Context-Free Grammar

- A grammar can be used to describe the possible hierarchical structure of a program.
- A context free grammar has 4 components:
  - A set of tokens, known as terminal symbols.
  - A set of nonterminals.
  - A set of productions where each production consists of a nonterminal, called the left side of the production, an arrow, and a sequence of tokens and/or nonterminals, called the right side of the production.
  - A designation of one of the nonterminals as the start symbol.
- The token strings that can be derived from the start symbol form the language defined by the grammar.

Example Grammar and Derivation

list $\rightarrow$ list + digit

list $\rightarrow$ list - digit

list $\rightarrow$ digit

digit $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

list => list+digit => list – digit + digit =>

digit – digit + digit => 9 – digit + digit =>

9 – 5 + digit => 9 – 5 + 2

Parse Trees

- A parse tree pictorially shows how the start symbol of a grammar derives a specific string in the language.
- Given a context free grammar, a parse tree is a tree with the following properties:
  - The root is labeled by the start symbol.
  - Each leaf is labeled by a token or by ε.
  - Each interior node is labeled by a nonterminal.
  - If A is the nonterminal labeling some interior node and X1, X2, ..., Xn are the labels of the children of that node from left to right, then A $\rightarrow$ X1X2...Xn is a production.

Ambiguous Grammars

- The leaves (tokens) of a parse tree read from left to right form a legal string in the language defined by the associated grammar.
- If a grammar can have more than one parse tree generating the same string of tokens, then the grammar is said to be ambiguous.
- For a grammar representing a programming language, we need to ensure that the grammar is unambiguous or there are additional rules to resolve the ambiguities.

string → string + string | string – string

string → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Precedence and Associativity

- Precedence determines which operator is applied first when different operators appear in an expression and parentheses do not explicitly indicate the order.
- Associativity is used to define the order of operations when there are multiple operators with the same precedence in an expression.
  - Left associativity means that (x op1 y) is applied first in the expression (x op1 y op2 z) when op1 and op2 have the same precedence.
  - Right associativity means that (y op2 z) is applied first in the expression (x op1 y op2 z) when op1 and op2 have the same precedence.

Converting Infix to Postfix

- If E is a variable or constant, then the postfix notation for E is E itself.
- If E is an expression of the form E1 op E2, where op is any binary operator, then the postfix notation for E is E1' E2' op, where E1' and E2' are the postfix notations for E1 and E2, respectively.
- If E is an expression of the form (E1), then the postfix notation for E1 is also the postfix notation for E.

```
(9-5)+2 ⇒ 95-2+  9-(5+2) ⇒ 952+-
```

Syntax-Directed Definition

- Uses a grammar to define the syntactic structure.
- Associates attributes with each grammar symbol.
- Associates semantic rules for computing the values of the attributes.

Translation Scheme

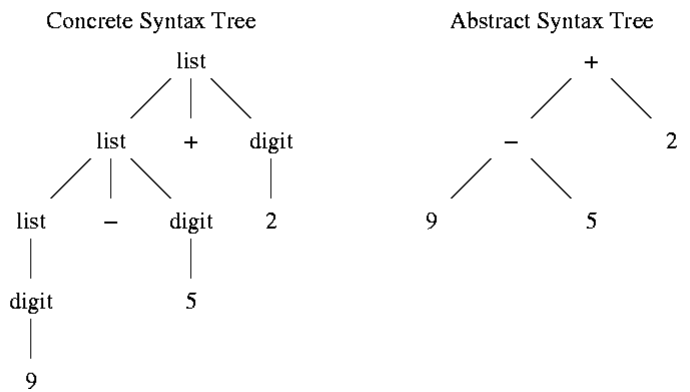
- A translation scheme is a grammar with program fragments called semantic actions that are embedded within the right hand side of the productions.
- Unlike a syntax-directed definition, the order of the evaluation of the semantic rules is explicitly shown.

Parsing

- Parsing is the process of determining if a string of tokens can be generated by a grammar.
- Parsing Methods
  - Top-Down
    - Construction starts at the root and proceeds to the leaves.
    - Can be more easily constructed by hand.
  - Bottom-Up
    - Construction starts at the leaves and proceeds to the root.
    - Can accept a larger class of grammars.

Syntax Trees

- Concrete Syntax Tree - a parse tree
- Abstract Syntax Tree
  - Each interior node is an operator rather than a nonterminal.
  - Convenient for translation.



Recursive Descent Parsing

- Top-down method for syntax analysis.
- A procedure is associated with each nonterminal of a grammar.
- Can be implemented by hand.
  - Decides which production to use by examining the lookahead symbol.
  - The appropriate procedure is invoked for each nonterminal in the rhs of the production.
- Predictive parsing means that a single lookahead symbol can be used to determine the procedure to be called for the next nonterminal.

Example Grammar for Recursive Descent Parsing

- Must not be left recursive.
- Must be left factored.

expr $\rightarrow$ term rest

rest $\rightarrow$ + term { print('+') } rest | - term { print('-') } rest | ε

term $\rightarrow$ 0 { print('0') }

term $\rightarrow$ 1 { print('1') }

...

term $\rightarrow$ 9 { print('9') }

Lexical Analysis Terms

- A token is a group of characters having a collective meaning.
  - id
- A lexeme is an actual character sequence forming a specific instance of a token.
  - `num`
- Characters between tokens are called whitespace.
  - blanks, tabs, newlines, comments

Buffering I/O

- It is too expensive to access a file one character at a time.
- Buffers are used for both input and output.
- Data is read from or written to the buffer until another buffer needs to be read or written.

Symbol Table

- Used to save lexemes (identifiers) and their attributes.
- It is common to initialize a symbol table to include reserved words so the form of an identifier can be handled in a uniform manner.
- Attributes are stored in the symbol table for later use in semantic checks and translation.

l-values and r-values

- l-value
  - Used on the left side of an assignment statement.
  - Used to refer to a location.
- r-value
  - Used on the right side of an assignment statement.
  - Used to refer to a value.

Abstract Stack Machine

- Stack machines are a common form used for the intermediate representation of a program.
  - push v — push v onto the stack
  - rvalue l — push contents of data location l
  - lvalue l — push address of data location l
  - pop — throw away value on top of the stack
  - := — the r-value on top is placed in the l-value below it and both are popped
  - copy — push a copy of the top value on the stack

Example of Stack Machine Intermediate Code

```
day := (1461*y) div 4 + (153*m + 2) div 5 + d;
```

```
lvalue day        push 2
push 1461         +
rvalue y          push 5
*                 div
push 4            +
div               rvalue d
push 153          +
rvalue m          :=
*
```

Control Flow

- Support for control flow is needed when translating statements.
  - label l — target of jumps to l; has no other effect
  - goto l — next instruction is taken from statement with label l
  - gofalse l — pop the top value; jump if it is zero
  - gotrue l — pop the top value; jump if it is nonzero
  - halt — stop execution