Module 10

Compiler Basics

Module Ten

- This week, we are going to talk about :

- The Compiler

  Language Definition - Grammar

  Lexical Analysis

  Syntactical Analysis

  Semantic Analysis

- Material adapted from L. Beck, System Software,1997
  A. Aho, Compliers, 1986

Basic Compiler Function

- Translate
  - Text written in one language
  - converted into a second language
- System Software
- High Level language
  - FORTRAN, ALGOL, Pascal, C, C++
  - Write solution to problem as engineer knows
- Machine code
  - Solution as executed by the hardware.

Basic Compiler Tasks

- Lexical Analysis - Scanner
  - Read the input text and find the token objects.
- Syntactical Analysis - Parser
  - From the tokens, discover the structure of the program
- Semantic Analysis - Code Generator
  - From the structure, create the machine language.

What is this ?

- IF THEN THEN THEN = ELSE ELSE ELSE = THEN

```
IF  (THEN) THEN
    THEN = ELSE;
ELSE
    ELSE = THEN;
```

A program

```
PROGRAM Stats

VAR
   sum, sumSQ, I, value, mean, variance : INTEGER

BEGIN
   sum    := 0;
   sumSQ  := 0;
   FOR I  := 1 TO 100 DO
        BEGIN
               READ ( value );
               sum    := sum + value;
               sumSQ  := sumSQ + value * value
        END;
   mean     := sum DIV 100;
   variance := sumSQ DIV 100 - mean * mean;
   WRITE ( mean, variance )
END.
```

Grammar

- Basic Language definition

- Formal description of the Programming Language
  - SYNTAX - form of legal statements
  - Structure - appropriate collection of the statements
  - Rules - 'how to' for constructing statements

- Example: ADD
  - I := J + K ; I, J, K, integers
  - Z := X + Y ; X, Y, Z, real

- The meanings of the statements is called Semantics.

Grammar

- BNF - Backus - Naur Form
  - Notation for writing a grammar. < symbol > **::=** tokens
  - Early 1960's
  - Used to define ALGOL
  - Top - Down set of rules for defining the objects within the programming language
  - Simple and widely used
  - Generally sufficient

Backus - Naur Form

- Set of Rules that define the syntax of a language object
- < symbol > **::=** tokens
- This < non-terminal symbol > is defined to be, this set of tokens
- Tokens - terminal symbol
- < symbol > **::=** < set of symbols or tokens >

A Simple Pascal Grammar

1. <prog> ::= PROGRAM <prog-name> VAR <dec-list> BEGIN <stmt-list> END.
2. <prog-name> ::= **id**

3. <dec-list> ::= <dec> | <dec-list> **,** <dec>
4. <dec> ::= <id-list> **:** <type>
5. <id-list> ::= **id** | <id-list> **,** **id**
6. <type> ::= INTEGER

7. <stmt-list> ::= <stmt> | <stmt-list> **;** <stmt>
8. <stmt> ::= <assign> | <read> | <write> | <for>

A Simple Pascal Grammar

9. <assign> ::= **id :=** <exp>
10. <exp> ::= <term> | <exp> **+** <term> | <exp> **-** <term>
11. <term> ::= <factor> | <term> * <factor> | <term> **DIV** <factor>
12. <factor> ::= **id** | **int** | **(** <exp> **)**
13. <read> ::= READ **(** <id-list> **)**
14. <write> ::= WRITE **(** <id-list> **)**
15. <for> ::= FOR <index-exp> DO <body>
16. <index-exp> ::= **id** := <exp> TO <exp>
17. <body> ::= <stmt> | BEGIN <stmt-list> END

BNF Examples

- <stmt> **::=** <assign> **|** <read> **|** <write> **|** <for>
- <read> **::=** READ **(** <id-list> **)**
- <id-list> **::=** **id** **|** <id-list> **,** **id**

Rule order

- Note the order of the rules.
- <exp> ::= <term> | <exp> **+** <term> | <exp> **-** <term>
- <term> ::= <factor> | <term> * <factor> | <term> **DIV** <factor>
- <factor> ::= id | int | **(** <exp> **)**

- Rule order assures that

  Q + 7 * A - 1

- Parses as if parentheses existed: Q + (7 * A) - 1

Grammar Review

- Formal definition of a programming Language
- Rules for forming the structures of the language.
- BNF notation
- Meanings

Summary

- Compiler Basics

  Language Definition - Grammar

  Next: Lexical Analysis