



Module 10

Compiler Basics



Module Ten

- Compiler Basics - Part Three
- In this presentation, we are going to talk about :
- Syntactical Analysis



Overview

- Previously we talked about:
- Compiler Basic Functions
- Language Definition – Grammar
- Lexical Analysis

Next: Syntactical Analysis



Syntactical Analysis

- Build the Structures.
- Recognize the constructs.
- Build the parse tree.
- BOTTOM_UP
 - Begin with the tokens and attempt to build a structure.
- TOP_DOWN
 - Begin with a goal and attempt to reach it.



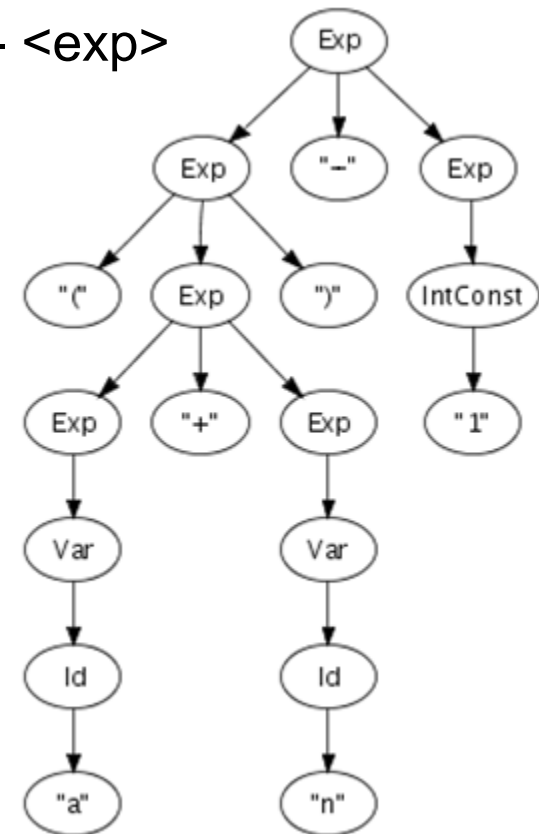
Parse tree

- Used to graphically display the analysis of the source statements.
- Shows the structure - Syntax Tree
- Given: Statements + grammar \Rightarrow Parse Tree
- Given: Parse Tree + grammar \Rightarrow Statements
- Diagram a Program.
- More than one tree, then ambiguous grammar.

Parse tree

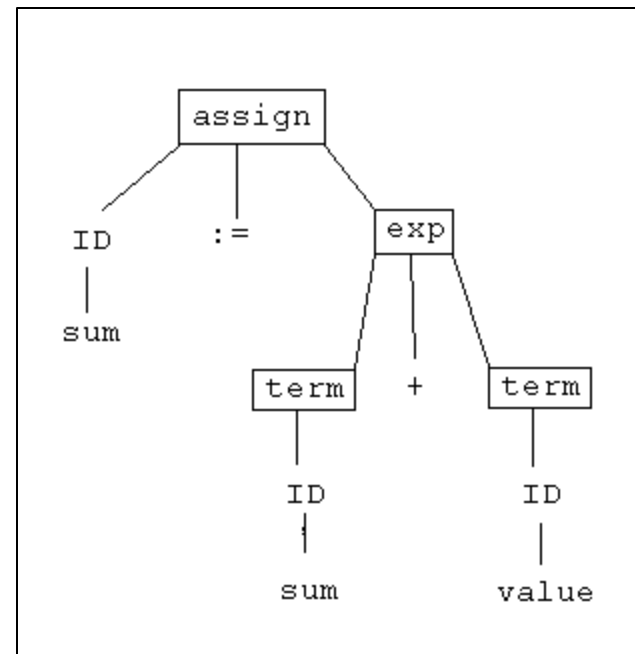
- Given this grammar:
- $\langle \text{exp} \rangle ::= \langle \text{exp} \rangle \mid \langle \text{exp} \rangle + \langle \text{exp} \rangle \mid \langle \text{exp} \rangle - \langle \text{exp} \rangle$
- $\langle \text{exp} \rangle ::= \text{id} \mid \text{int} \mid (\langle \text{exp} \rangle)$

- $(a + n) - 1$



Parse tree

- Given this grammar:
- $\langle \text{assign} \rangle ::= \text{id} := \langle \text{exp} \rangle$
- $\langle \text{exp} \rangle ::= \langle \text{term} \rangle \{ + \langle \text{term} \rangle \mid - \langle \text{term} \rangle \}$
- and this statement
- sum := sum + value;**



Syntactical Analysis BOTTOM_UP

- **Operator Precedence Parsing**
- Operators are the Terminal Symbols of the language.
- Precedence Matrix
- Build (or obtain) the precedence matrix.
- Automatic generation.

Precedence Matrix

A + B * C - D

id1 + id2 *

	VAR	BEGIN	END	END.	INTEGER	FOR	READ	WRITE	TO	DO	;	:	,	:=	+	-	*	DIV	()	id	int
PROGRAM	≡																				^	
VAR	≡									^	^	^									^	^
BEGIN		≡	≡		^	^	^			^											^	
END		∇	∇							∇												
INTEGER	∇									∇											^	
FOR									≡												^	
READ																		≡				
WRITE																		≡				
TO									∇					^	^	^	^	^	^	^	^	^
DO	^	∇	∇		^	^	^	^		∇											^	^
;	∇	∇	∇		^	^	^	^		∇	^	^									^	^
:	∇				^					∇											^	^
,																				≡		
:=		∇	∇						≡	∇	∇				^	^	^	^	^	^	^	^
+		∇	∇						∇	∇	∇				∇	∇	∇	∇	∇	∇	∇	∇
-		∇	∇						∇	∇	∇				∇	∇	∇	∇	∇	∇	∇	∇
*		∇	∇						∇	∇	∇				∇	∇	∇	∇	∇	∇	∇	∇
DIV		∇	∇						∇	∇	∇				∇	∇	∇	∇	∇	∇	∇	∇
(^			^	^	^	^	^	≡	^	^
)		∇	∇						∇	∇	∇				∇	∇	∇	∇	∇	∇		
id	∇	∇	∇						∇	∇	∇	∇	∇	≡	∇	∇	∇	∇	∇	∇		
int		∇	∇						∇	∇	∇				∇	∇	∇	∇	∇	∇		

Figure 5.11 Precedence matrix for the grammar from Fig. 5.2.

Syntactical Analysis BOTTOM_UP

- **Operator Precedence Parsing**
- Scan for subexpressions having operators with higher precedence than the surrounding operators.
- $\langle \cdot \quad \cdot \quad \cdot \quad \cdot \rangle$
- Scan left to right, only as many tokens as needed.

A + B * C - D

$\langle \cdot \quad \cdot \quad \cdot \quad \cdot \rangle$

$\langle \cdot \quad \cdot \quad \cdot \quad \cdot \rangle$

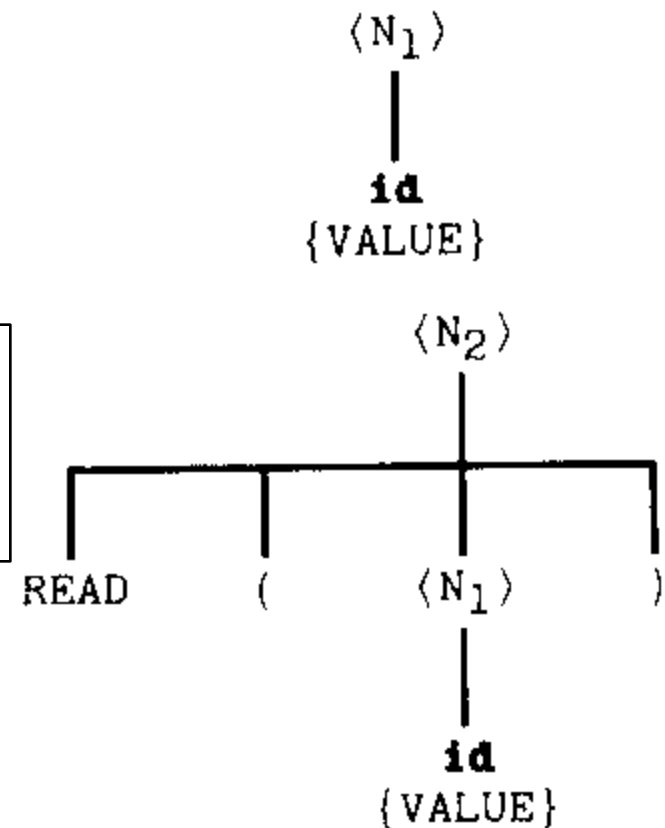
Bottom - Up

- Operator-precedence parse of a READ statement
- ... BEGIN READ (VALUE) ;

(i) ... BEGIN READ (**id**)
 $\leq \quad \quad \quad \doteq \quad \leq \quad \geq$

(ii) ... BEGIN READ ($\langle N_1 \rangle$) ;
 $\leq \quad \quad \quad \doteq \quad \quad \quad \doteq \quad \geq$

(iii) ... BEGIN $\langle N_2 \rangle$;
 \leq





Syntactical Analysis BOTTOM_UP

- **Shift - Reduce Parsing**
- Shift unrecognized tokens onto stack.
- When recognized, Reduce the stack and place non-terminal onto stack.
- Can be applied to a class of grammars known as LR.
(Left-right scan, Reverse derivation)
- Symbols to be recognized always at top of stack.
- LR(k) Grammars where k is the token lookahead count.

Syntactical Analysis TOP_DOWN

- **Recursive Descent Parsing**
- Procedure for each non-terminal symbol in the Grammar.
- Procedures attempt to find the substring of the input that satisfies the non-terminal symbol.
- May call other procedures, even itself.
- If procedure finds the non-terminal, it advances input token pointer and returns success.
- Otherwise, it returns failure, and / or calls the Error Routine.



ASSIGN procedure

Procedure **ASSIGN**

<assign> ::= id := <exp>

```
begin
    FOUND := FALSE
    if TOKEN = id then
        begin
            get next TOKEN
            if TOKEN = := then
                begin
                    get next TOKEN
                    if EXP returns SUCCESS then
                        FOUND := TRUE
                    end if :=
                end if id
            if FOUND = TRUE
                return SUCCESS
            else
                return FAILURE
            end if id
        end if id
    end if id
end ASSIGN
```



Syntactical Analysis TOP_DOWN

- **Recursive Descent Parsing**
- Grammar needs to be defined to eliminate 'Left Recursion'
 - $\langle \text{id-list} \rangle ::= \text{id} \mid \langle \text{id-list} \rangle , \text{id}$
 - $\langle \text{id-list} \rangle ::= \text{id} \{ , \text{id} \}$
- Read the next token to determine path among alternatives.
- Some compilers use both TOP_DOWN and BOTTOM_UP parsing.

Revised Grammar

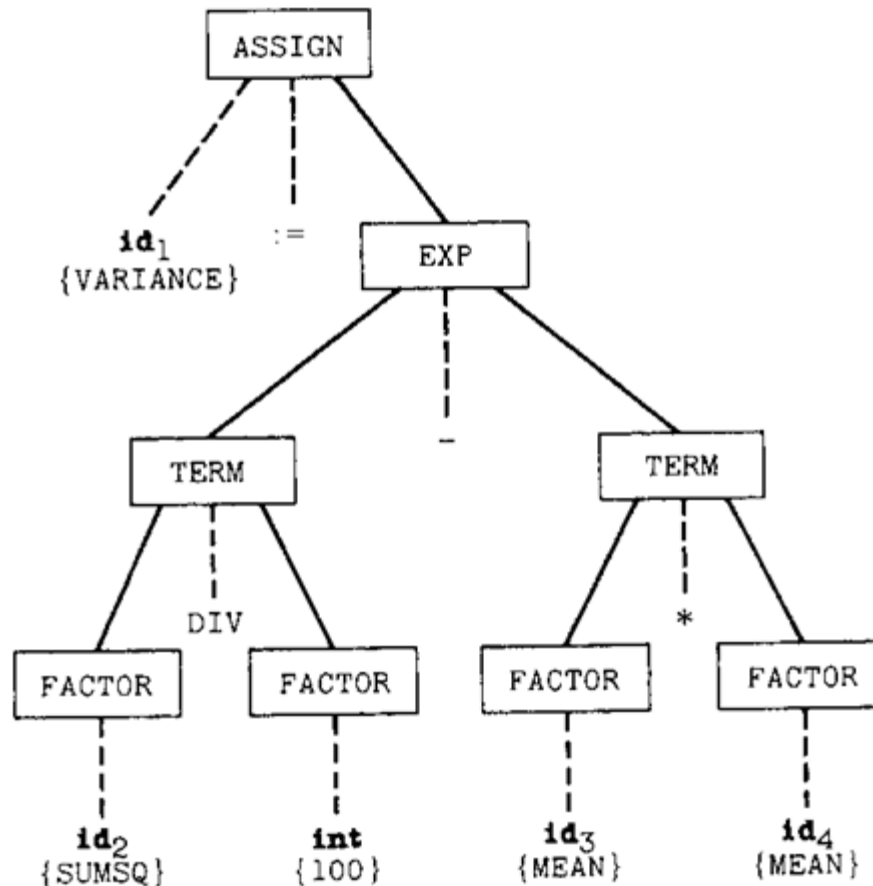
Recursive Descent

- $\langle \text{prog} \rangle ::= \text{Program } \langle \text{prog-name} \rangle \text{ VAR } \langle \text{dec-list} \rangle \text{ BEGIN } \langle \text{stmt-list} \rangle \text{ END.}$
- $\langle \text{prog-name} \rangle ::= \text{id}$
- $\langle \text{dec-list} \rangle ::= \langle \text{dec} \rangle \{ ; \langle \text{dec} \rangle \}$
- $\langle \text{dec} \rangle ::= \langle \text{id-list} \rangle : \langle \text{type} \rangle$
- $\langle \text{type} \rangle ::= \text{INTEGER}$
- $\langle \text{id-list} \rangle ::= \text{id} \{ , \text{id} \}$
- $\langle \text{stmt-list} \rangle ::= \langle \text{stmt} \rangle \{ ; \langle \text{stmt} \rangle \}$
- $\langle \text{stmt} \rangle ::= \langle \text{assign} \rangle \mid \langle \text{read} \rangle \mid \langle \text{write} \rangle \mid \langle \text{for} \rangle$
- $\langle \text{assign} \rangle ::= \text{id} := \langle \text{exp} \rangle$
- $\langle \text{exp} \rangle ::= \langle \text{term} \rangle \{ + \langle \text{term} \rangle \mid - \langle \text{term} \rangle \}$
- $\langle \text{term} \rangle ::= \langle \text{factor} \rangle \{ * \langle \text{factor} \rangle \mid \text{DIV } \langle \text{factor} \rangle \}$
- $\langle \text{factor} \rangle ::= \text{id} \mid \text{int} \mid (\langle \text{exp} \rangle)$
- $\langle \text{read} \rangle ::= \text{READ } (\langle \text{id-list} \rangle)$
- $\langle \text{write} \rangle ::= \text{WRITE } (\langle \text{id-list} \rangle)$
- $\langle \text{for} \rangle ::= \text{FOR } \langle \text{index-exp} \rangle \text{ DO } \langle \text{body} \rangle$
- $\langle \text{index-exp} \rangle ::= \text{id} := \langle \text{exp} \rangle \text{ TO } \langle \text{exp} \rangle$
- $\langle \text{body} \rangle ::= \langle \text{stmt} \rangle \mid \text{begin } \langle \text{stmt-list} \rangle \text{ END}$

Top-Down

- Recursive-descent parsing of an ASSIGN statement

VARIANCE := SUMSQ DIV 100 - MEAN * MEAN





Summary

- Syntax Analysis
 - Parsing
 - Bottom Up
 - Top Down

Next: Semantic Analysis