

What a compiler does?

Stack-based Machine:

```
float x;  
int main()  
{  
    int i;  
    x = 123;  
    i++;  
}
```

```
Program(1, 3);  
Variable(1, 3);  
Constant(123);  
Assignment(1);  
Variable(0, 3);  
Variable(0, 3);  
Value(1);  
Constant(1);  
Add;  
Assignment(1);  
EndProgram;
```

Register-based Machine:

Rule 1.1 A compiler must be error-free.

Rule 1.2 A compiler must always terminate, no matter what the input looks like.

Rule 1.3 A compiler should attempt to find as many errors as possible during a single compilation.

Lexical Analysis

Syntax Analysis

Error Recovery

Scope Analysis

Type Analysis

Code Generation

Errors: lexical, syntax, semantic and logical errors

Vocabulary (tokens)

- 1) word symbols (reserved words)
- 2) names (identifiers)
- 3) numerals (constants)
- 4) special symbols (operators)

Pascal- Grammar (Backus-Naur Form or Context-free Grammar)

1. Program = “program” ProgramName “;” BlockBody “.”
2. BlockBody =
 [ConstantDefinitionPart] [TypeDefinitionPart] [VariableDefinitionPart]
 { ProcedureDefinition } CompoundStatement
3. ConstantDefinitionPart =
 “const” ConstantDefinition { ConstantDefinition }
4. ConstantDefinition = ConstantName “=” Constant “;”
5. TypeDefinitionPart =
 “type” TypeDefinition { TypeDefinition }
6. TypeDefinition = TypeName “=” NewType “;”
7. NewType = NewArrayType | NewRecordType
8. NewArrayType =
 “array” “[“ IndexRange “]” “of” TypeName
9. IndexRange = Constant “..” Constant
10. NewRecordType = “record” FieldList “end”
11. FieldList = RecordSection { “;” RecordSection }
12. RecordSection =
 FieldName { “;” FieldName } “:” TypeName
13. VariableDefinitionPart =
 “var” VariableDefinition { VariableDefinition }
14. VariableDefinition = VariableGroup “;”
15. VariableGroup =
 VariableName { “;” VariableName } “:” TypeName
16. ProcedureDefinition =
 “procedure” ProcedureName ProcedureBlock “;”
17. ProcedureBlock =
 [“(“ FormalParameterList “)”] “;” BlockBody
18. FormalParameterList =
 ParameterDefinition { “;” ParameterDefinition }
19. ParameterDefinition = [“var”] VariableGroup
20. Statement =
 AssignmentStatement | ProcedureStatement | IfStatement | WhileStatement |
 CompoundStatement | Empty
21. AssignmentStatement =
 VariableAccess “:=” Expression
22. ProcedureStatement =
 ProcedureName [“(“ ActualParameterList “)”]
23. ActualParameterList =
 ActualParameter { “;” ActualParameter }
24. ActualParameter = Expression | VariableAccess
25. IfStatement =
 “if” Expression “then” Statement [“else” Statement]
26. WhileStatement = “while” Expression “do” Statement
27. CompoundStatement = “begin” Statement { “;” Statement } “end”

28. Expression = SimpleExpression [RelationalOperator SimpleExpression]
29. RelationalOperator = "<" | "=" | ">" | "<=" | "<" | ">="
30. SimpleExpression = [SignOperator] Term {AddingOperator Term}
31. SignOperator = "+" | "-"
32. AddingOperator = "+" | "-" | "or"
33. Term = Factor {MultiplyingOperator Factor}
34. MultiplyingOperator = "*" | "div" | "mod" | "and"
35. Factor = Constant | VariableAccess | "(" Expression ")" | "not" Factor
36. VariableAccess = VariableName {Selector}
37. Selector = IndexedSelector | FieldSelector
38. IndexedSelector = "[" Expression "]"
39. FieldSelector = "." FieldName
40. Constant = Numeral | ConstantName
41. Numeral = Digit {Digit}
42. Name = Letter {Letter | Digit}