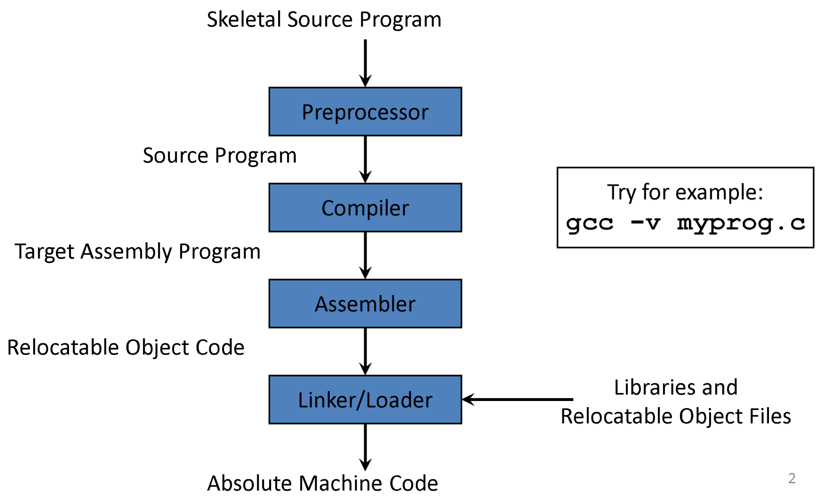
Compiler Construction

Chapter-1

Language Processing System



Try for example:

```
gcc -v myprog.c
```

2

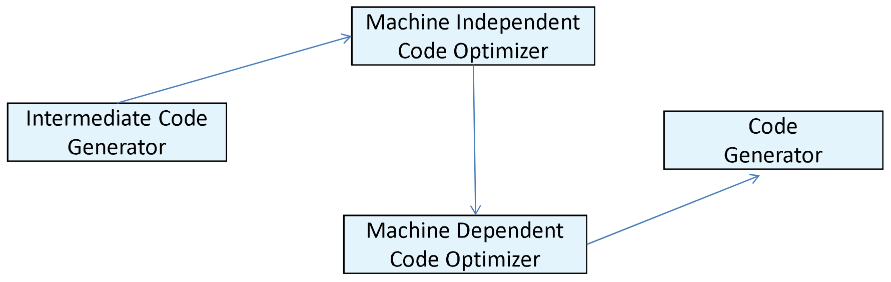
Analysis Part of Compilation

- Three Phases:
  - Linear / Lexical Analysis:
    - L-to-R Scan to Identify Tokens

      token: sequence of chars having a collective meaning
  - Hierarchical Analysis (Syntax Analysis):
    - Grouping of Tokens Into Meaningful Collection
  - Semantic Analysis:
    - Checking to ensure Correctness of Components

Synthesis Part of Compilation

- Code Optimizer (Optional Phase)
  - Machine Independent Code Optimizer
  - Machine Dependent Code Optimizer
- Code Generation


Machine Independent Code Optimizer
Intermediate Code Generator
Code Generator
Machine Dependent Code Optimizer

Other Tools that Use the Analysis-Synthesis Model

- *Editors* (syntax highlighting)
- *Pretty printers* (e.g. Doxygen)
- *Static checkers* (e.g. Lint and Splint)
- *Text formatters* (e.g. TeX and LaTeX)
- *Silicon compilers* (e.g. VHDL)
- *Query interpreters/compilers* (Databases)

5

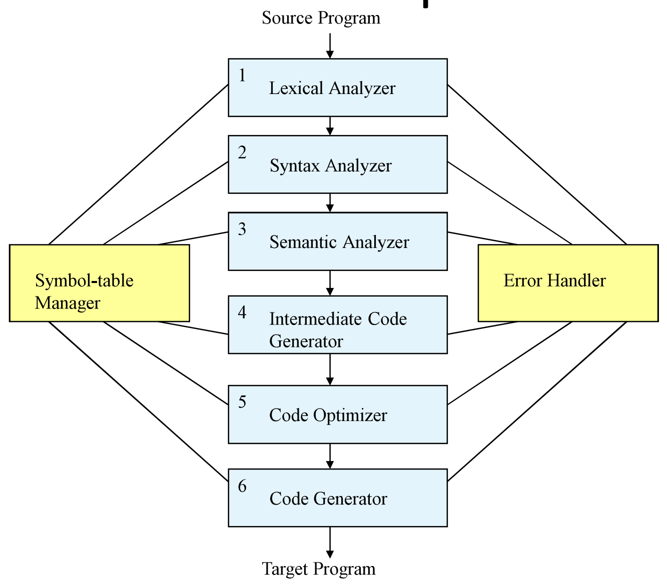
Other Tools that Use the Analysis-Synthesis Model

- **Pretty Printers:** Standardized version for program structure (i.e., blank space, indenting, etc.)
  - Analyzes the source program and prints it in such a way that the structure of the program becomes clearly visible.
  - Examples (Doxygen)
    - Comments may appear in a special font
    - Statements may appear with an amount of indentations proportional to the depth of their nesting in a hierarchical organization of the stmts.
- **Static Checkers:** A “quick” compilation to detect rudimentary errors
  - Examples (Lint & Splint)
    - Detects parts of the program that can never be executed
    - A variable used before it is defined

Other Tools that Use the Analysis-Synthesis Model

- Text Formatters
  - LATEX & TROFF Are Languages Whose Commands Format Text (paragraphs, figures, mathematical structures etc)
- Silicon Compilers (VHDL)
  - Textual / Graphical: Take Input and Generate Circuit Design
- Database Query Processors
  - Database Query Languages Are Also a Programming Language
  - Input is compiled Into a Set of Operations for Accessing the Database

Phases of Compiler



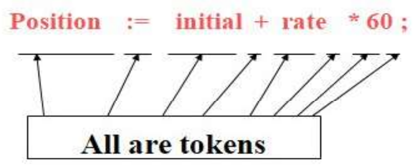
Supporting Tasks

- Symbol Table Creation / Maintenance
  - Contains Info (storage, type, scope, arguments) on Each “Meaningful” Token, Typically Identifiers
  - Data Structure Created / Initialized During Lexical Analysis
  - Utilized / Updated During Later Analysis & Synthesis
- Error Handling
  - Detection of Different Errors Which Correspond to All Phases
  - What Kinds of Errors Are Found During the Analysis Phase?
  - What Happens When an Error Is Found?

Lexical Analysis

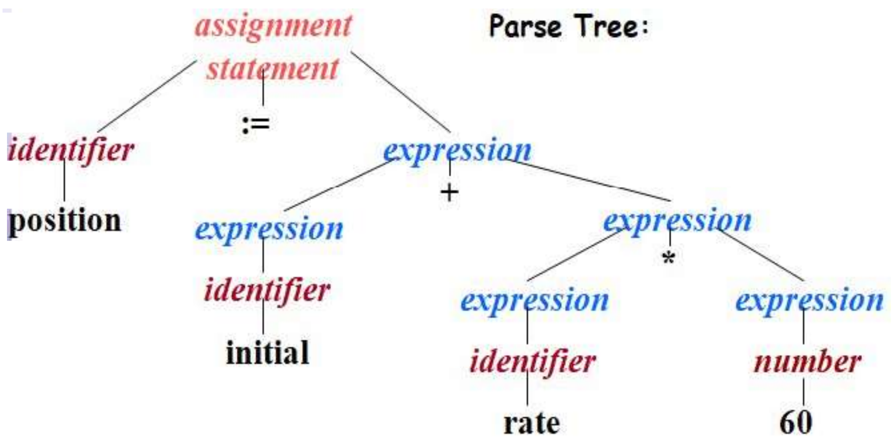
- Identify the **tokens** which are the basic building blocks

For Example:



Blanks, Line breaks, etc. are scanned out

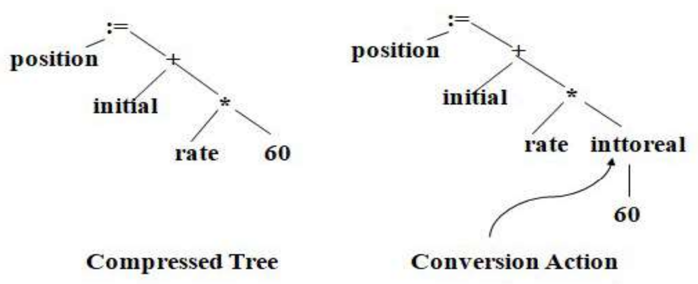
Syntax Analysis (Hierarchical Analysis)



Semantic Analysis (

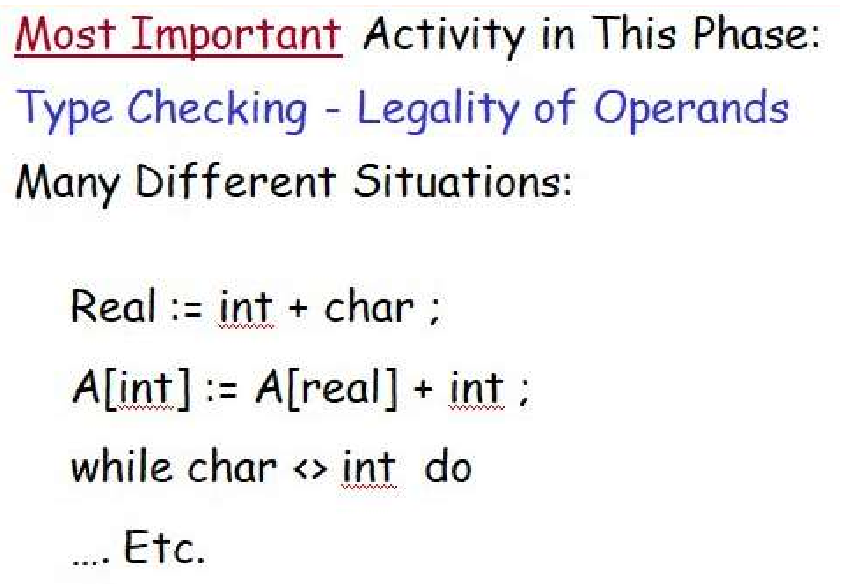
Find More Complicated Semantic Errors and Support Code Generation

Parse Tree Is Augmented With Semantic Actions



This phase generates : Annotated Parse Tree or Abstract Syntax Tree

Semantic Analysis



Most Important Activity in This Phase:

Type Checking - Legality of Operands

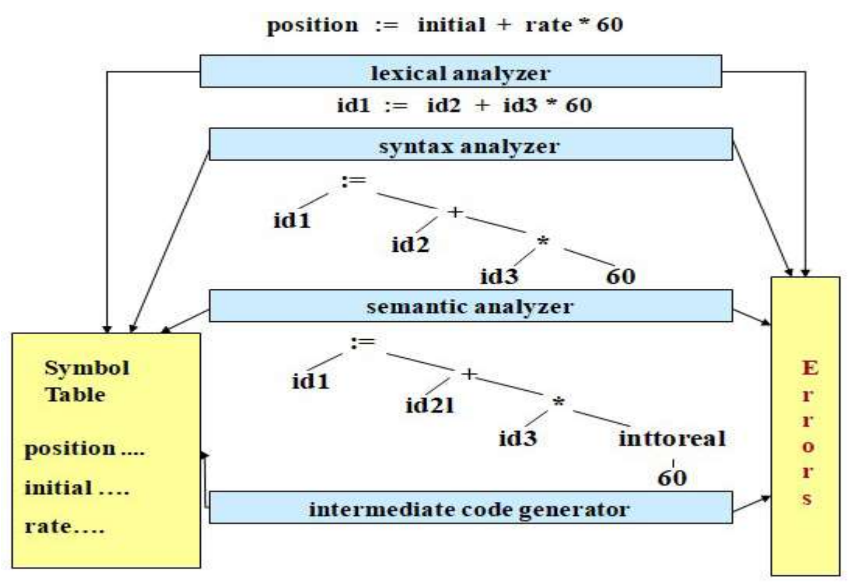
Many Different Situations:

```
Real := int + char ;
A[int] := A[real] + int ;
while char <> int  do
.... Etc.
```

Why is Analysis divided in this way ?

- Lexical Analysis - Scans Input, Its Linear Actions Are Not Recursive
  - Identify Only Individual “words” that are the Tokens of the Language
- Recursion Is Required to Identify Structure of an Expression, As Indicated in Parse Tree
  - Verify that the “words” are Correctly Assembled into “sentences”
- Semantic Analysis
  - Determine Whether the Sentences have One and Only One Unambiguous Interpretation

Review: Analysis



Intermediate Code Generator

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

3 address code

3-Address Code :

1. Each Three address assignment instruction has at most one operator on the right side.
2. Compiler must generate a temporary name to hold the value computed by a three-address instruction
3. Three address Instructions may have fewer than three operands

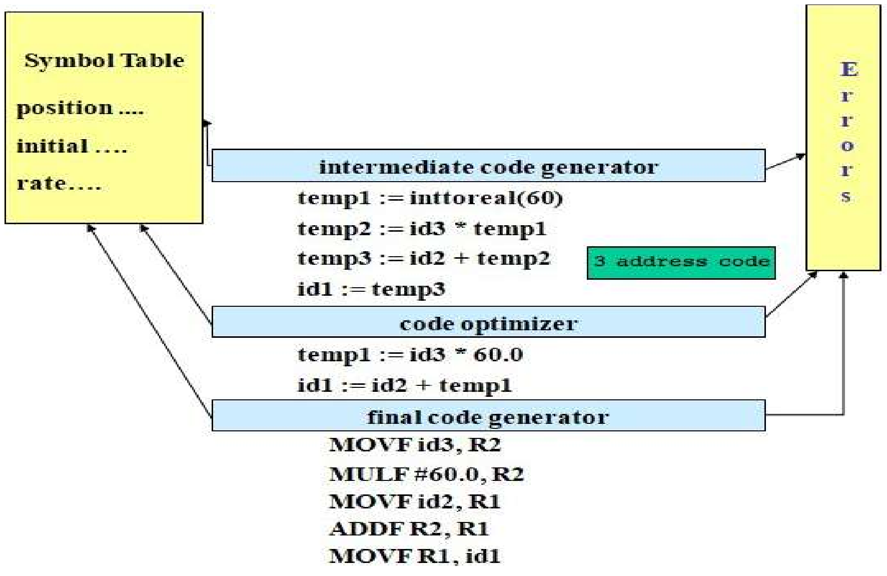
Code Optimizer

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

There are many MACHINE INDEPENDENT optimization techniques

1. Constant Folding
2. Dead Code Elimination
3. Frequency Reduction Optimization
4. Strength Reduction
5. Copy Propagation
6. Loop-invariant Code Motion
7. Common Sub-expression Elimination
8. Value Numbering

Review: Synthesis



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

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

```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

The Grouping of Phases

- Compiler *front* and *back ends*:
  - Front end: *analysis* (*machine independent*)
  - Back end: *synthesis* (*machine dependent*)
- Compiler *passes:*
  - A collection of phases is done only once (*single pass*) or multiple times (*multi pass*)
    - **Single pass**: usually requires everything to be defined before being used in source program. It takes more space. It is preferred for computers having large memory. It is very fast.
    - **Multi pass**: compiler may have to keep entire program representation in memory. It takes less space. It is preferred for computers having small memory. It is slow.

19

Compiler-Construction Tools

- Software development tools are available to implement one or more compiler phases
  - *Scanner generators*
  - *Parser generators*
  - *Syntax-directed translation engines*
  - *Automatic code generators*
  - *Data-flow engines*

20