UCS1602:
COMPILER DESIGN

Language processor

&

Phases of compiler

Session Objectives

- Learning about the language processors
- Understanding the concepts of phases of compiler

Session Outcomes

- At the end of this session, participants will be able to
  - Understand the language processors
  - Understand the phases of compiler

Agenda

- Introduction
- Language processor
- Phases of compiler

Course outline

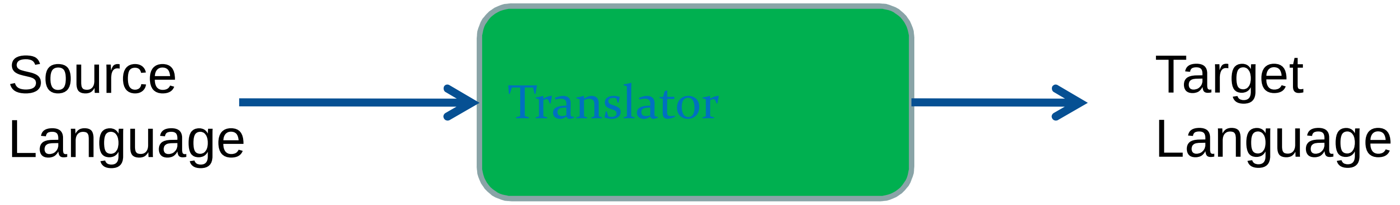
- Unit 1 – Introduction to compilers, Phases of compiler
  lexical analyser
- Unit 2 – Syntax analyser -

Course outline

- Introduction to Compiling
- Lexical Analysis
- Syntax Analysis
  - Context Free Grammars
  - Top-Down Parsing, LL Parsing
  - Bottom-Up Parsing, LR Parsing
- Syntax-Directed Translation
- Run-Time Organization
- Intermediate Code Generation
- Code Optimization
- Code Generation

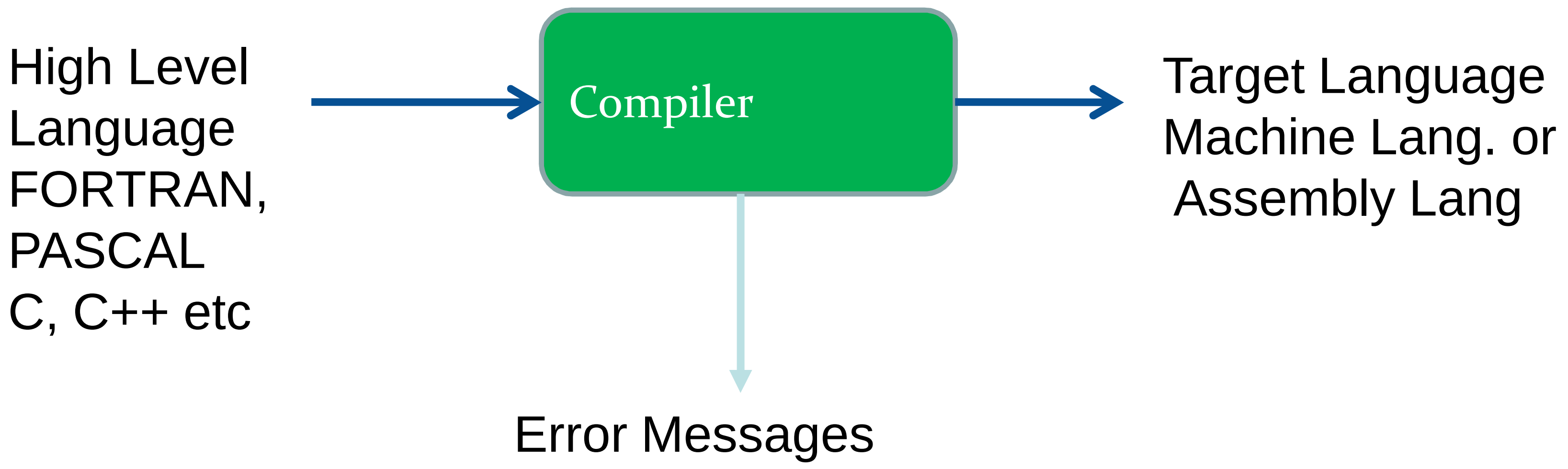
Language processors

- **Translators**



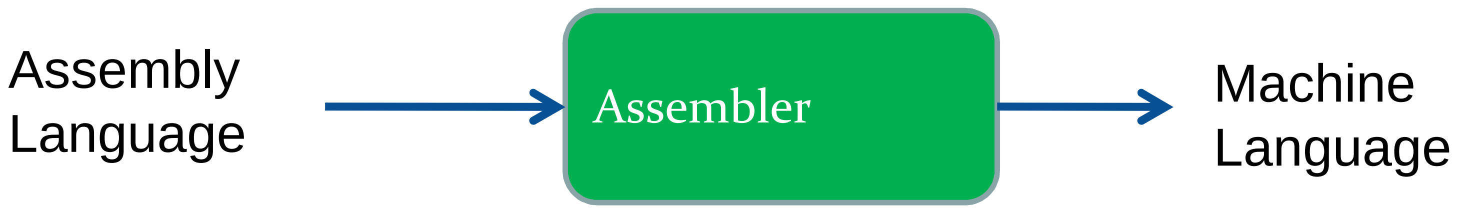
Language processors

- **Compiler**


High Level Language FORTRAN, PASCAL C, C++ etc
Compiler
Target Language Machine Lang. or Assembly Lang
Error Messages

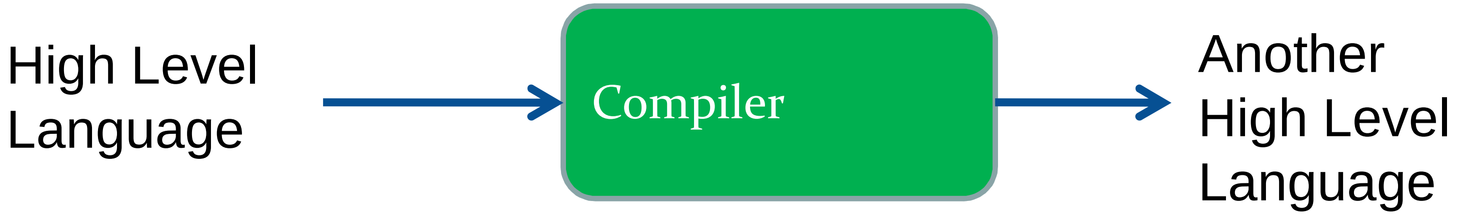
Language processors

- **Assembler**



Introduction to Compiling

- **Preprocessor**



Language processors

- Interpreter
  - Input → Intermediate code
  - Instead of output, performs operations implied by the source program.

Language processors

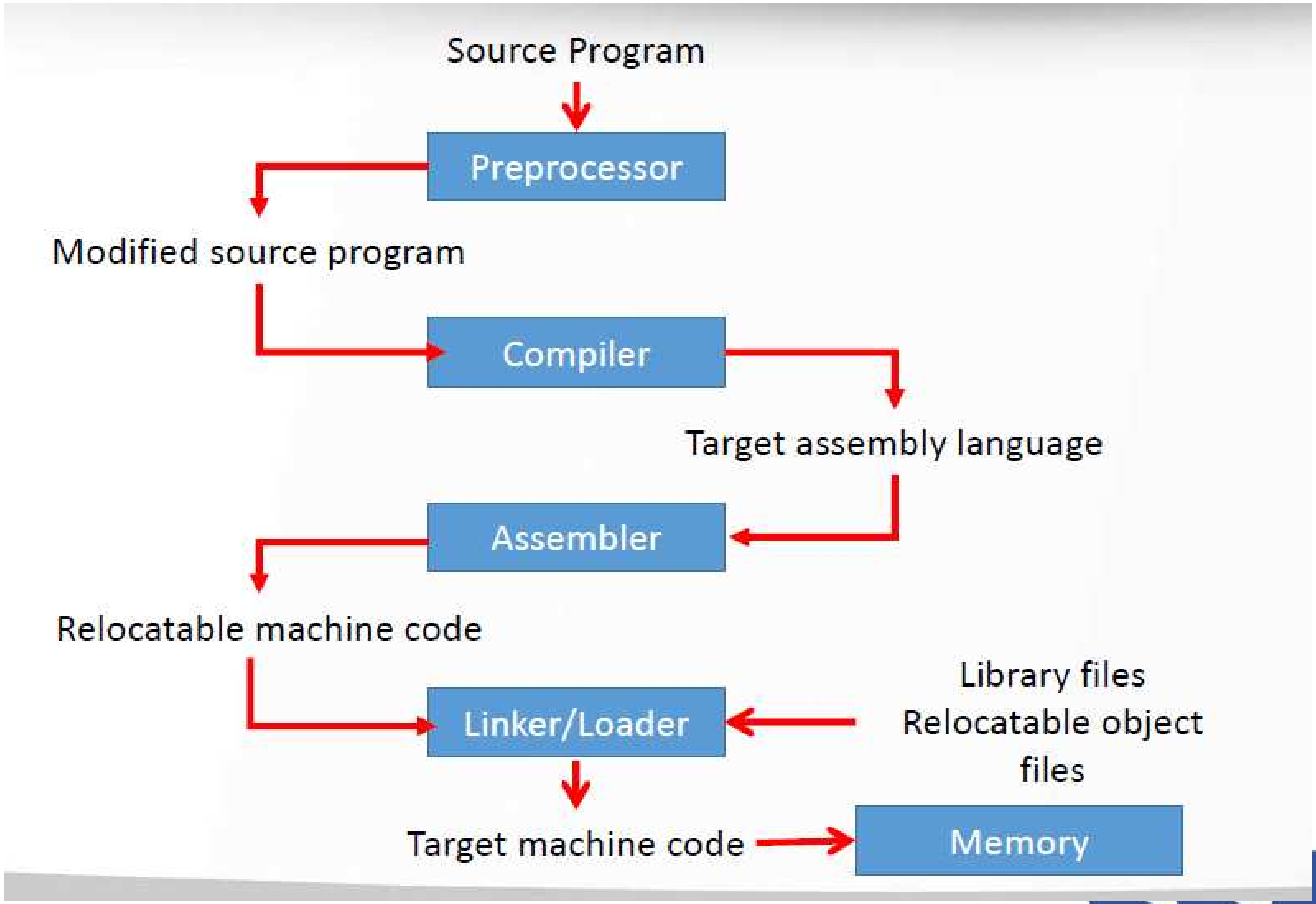
- Linker:

Linker is a computer program that links and merges various object files together in order to make an executable file.

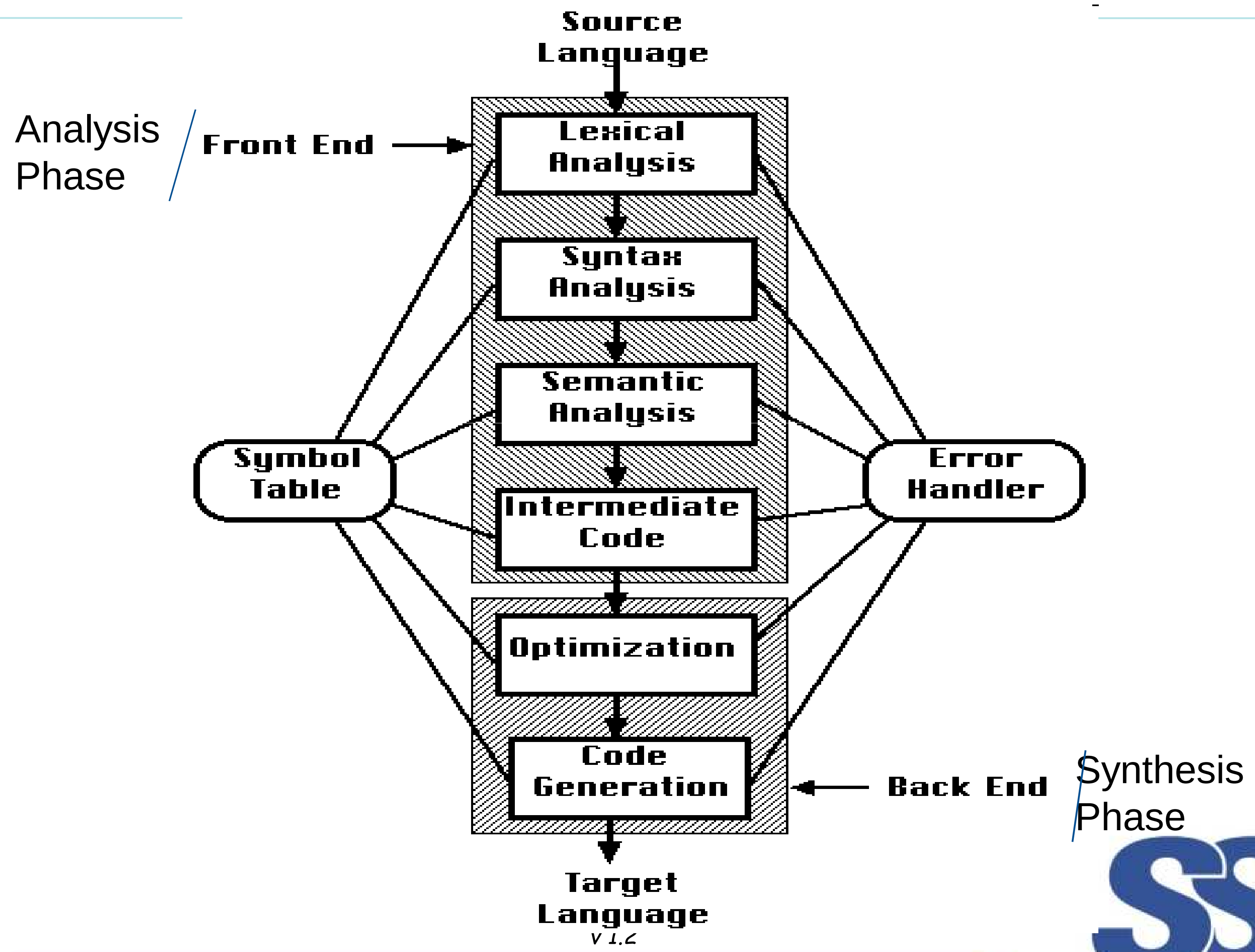
- Loader:

Loader is a part of operating system and is responsible for loading executable files into memory and execute them. It calculates the size of a program (instructions and data) and creates memory space for it. It initializes various registers to initiate execution.

Language processing phases



Phases of a Compiler



Major parts of Compiler

- **Analysis Phase** and **Synthesis Phase**
- **Analysis phase**
  - an intermediate representation is created from the given source program.
  - Lexical Analyzer, Syntax Analyzer, Semantic Analyzer and Intermediate Code Generator are the parts of this phase.
- **Synthesis phase**
  - the equivalent target program is created from this intermediate representation.
  - Code Generator, and Code Optimizer are the parts of this phase.

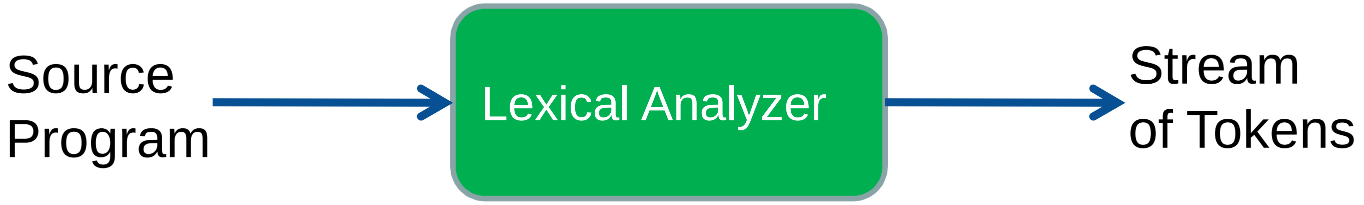
Applications

- Techniques used in a **lexical analyzer** can be used in **text editors, information retrieval system, and pattern recognition programs.**
- Parser $\rightarrow$ query processing system such as SQL.
- Many software having a complex front-end may need techniques used in compiler design.
- Most of the techniques used in compiler design can be used in Natural Language Processing (NLP) systems.

Overview of Compilation

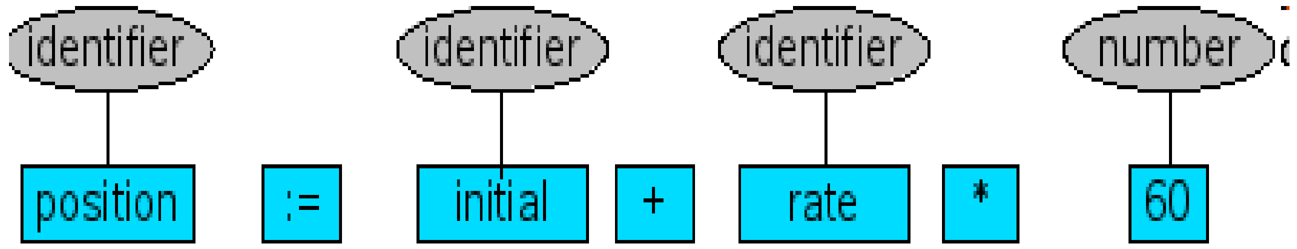
Lexical Analyzer

- Reads Characters from left to right
- Tokens → identifiers, keywords, operators, punctuation symbols, multi character operators



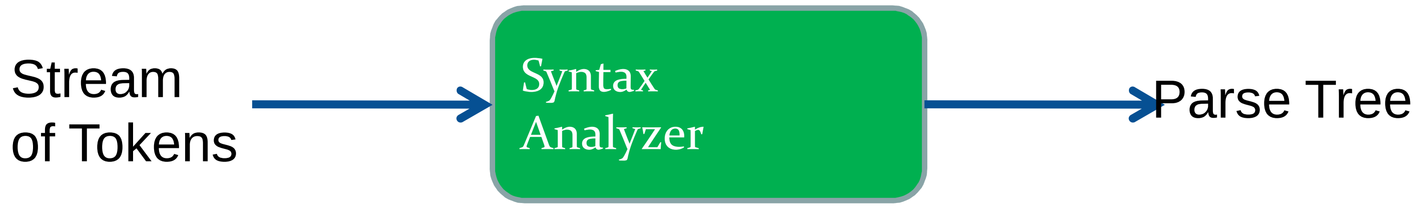
Lexical Analyzer Cont…

- position := initial + rate * 60
  - The identifier position.
  - The assignment symbol :=.
  - The identifier initial.
  - The plus sign.
  - The identifier rate.
  - The multiplication sign.
  - The number 60.



Syntax Analyzer

- Parser also know as Hierarchical analysis



- Performs 2 functions
  - Checks the token if they occur in patterns specified by the source language
  - Construct a tree like structure that can be used by the subsequent phases

Syntax Analyzer Cont…

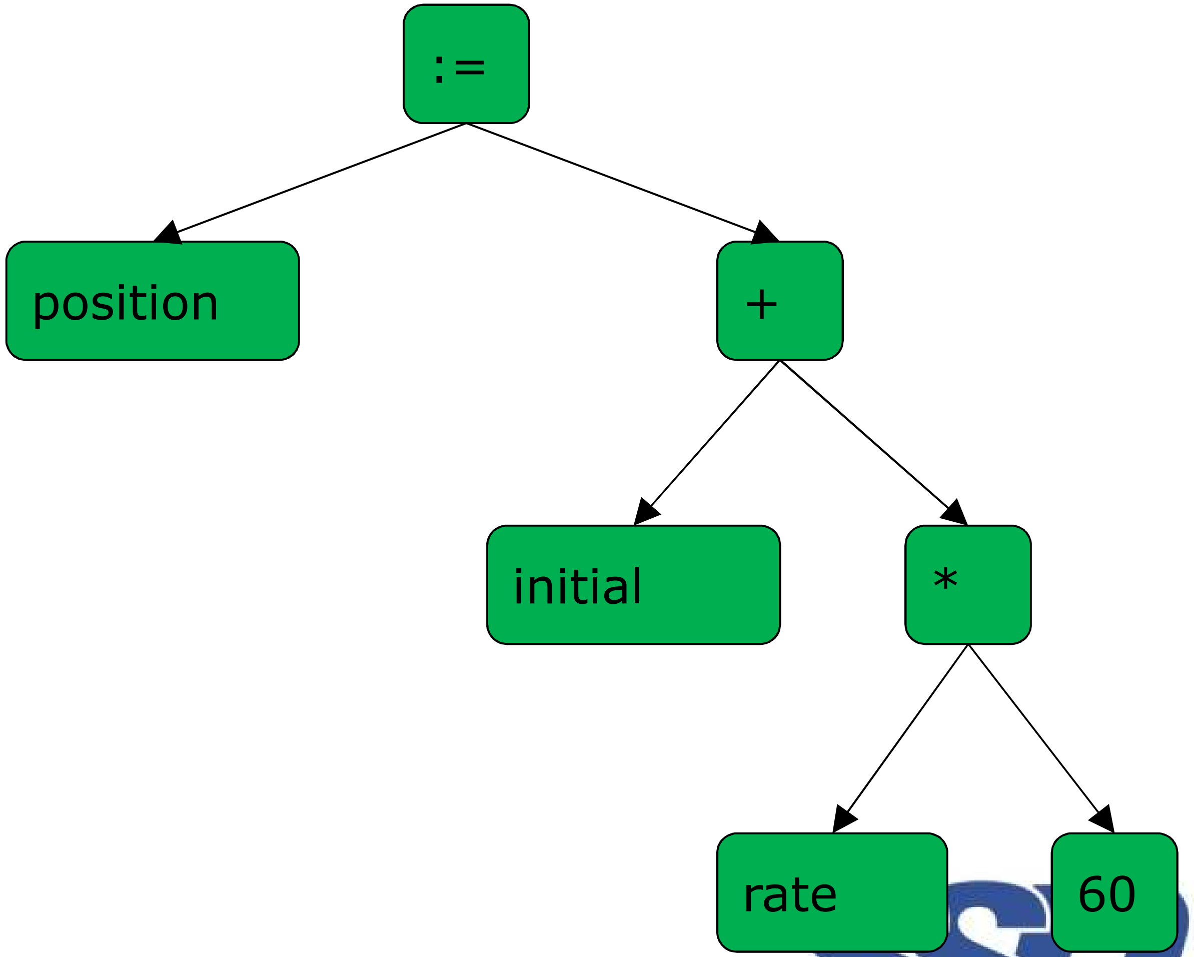
The hierarchical structure of programs is usually described by recursive rules like the following ones that describe expressions:

1. Any *Identifier* is an *expression*.
2. Any *Number* is an *expression*.
3. If *Expression*1 and *Expression*2 are expressions, then so are:
    a. *Expression*1+*Expression*2
    b. *Expression*1**Expression*2
    c. (*Expression*1)
4. If *Identifier*1 is an identifier and *Expression*2 is an expression, then
   *Identifier*1 := *Expression*2   is a statement.
5. If *Expression*1 is an expression and *Statement*2 is a statement, then
    a. while (*Expression*1) do*Statement*2
    b. if (*Expression*1) then*Statement*2   are statements.

Syntax Analyzer Cont…

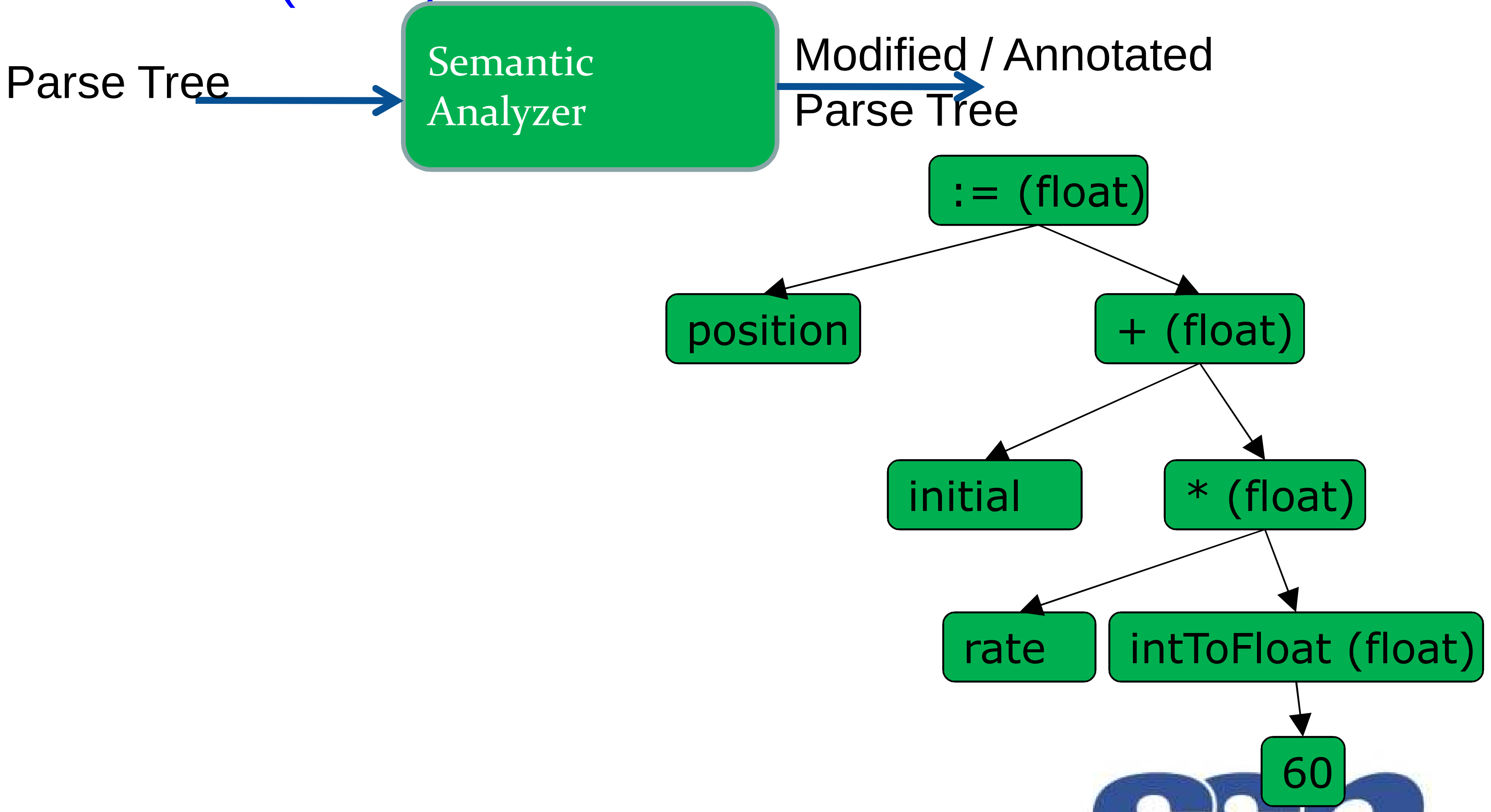
- According to rule (1)
- position, initial and rate are expressions.
- Rule (2) states that 60 in an expression.
- Rule (3) says that rate * 60 is an expression
- Finally Rule (4) says that initial + rate * 60 is an expression.

Syntax Analyzer Cont…



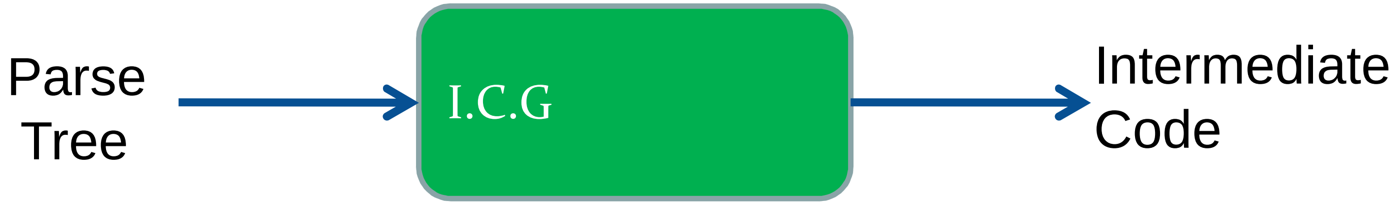
Semantic Analyzer

- Checks for (more) "static semantic" errors



Intermediate Code Generator

- Variety of Intermediate Representations



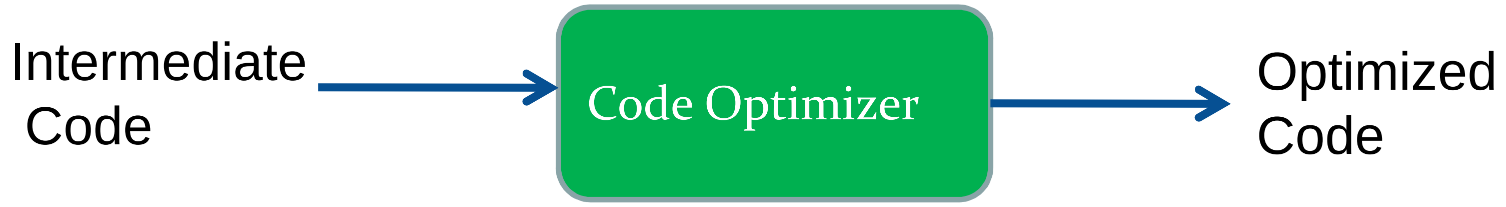
```
temp1 := inttofloat(60)
temp2 := rate * temp1
temp3 := initial + temp2
position := temp3
```

Intermediate Code Generator

- Properties of 3 Address Code
  - Atmost one operator other than assignment operator.
  - Compiler must generate a temporary name to hold the value computed by each instruction.
  - Some 3 address instruction can have less than 3 operands.

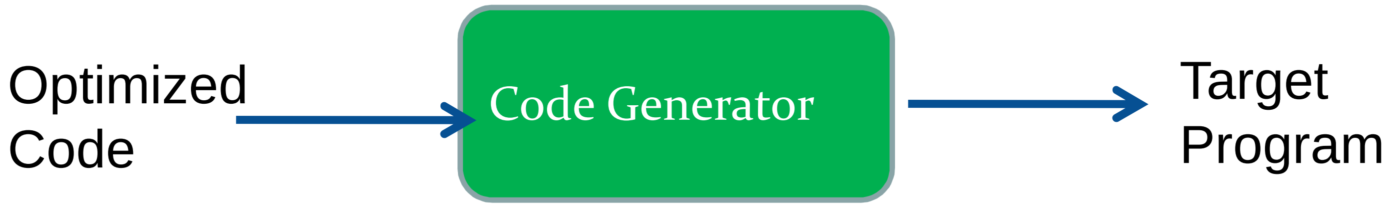
Code Optimizer

- Tries to improve code to
  - Run faster
  - Be smaller
  - Consume less energy



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

Code Generator



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

Symbol Table

- Keep track of names declared in the program
- Separate level for each scope
- Linear List → Slow but easy to implement
- Hash table → Complex to implement but fast.

Error Handler

- Involves
  - Detection of errors
  - Reporting Errors
  - Recovery of Errors.

Summary

- Introduction
- Language processors
- Phases of compiler
- Symbol table
- Error handling

Check your understanding?

1. What is compiler?
2. What is assembler?
3. Two major parts of the compiler are _____________ and____________
4. List the applications of compiler.
5. Lexical analyser is otherwise called as ______________
6. Syntax analyser is otherwise called as ______________
7. The data structures used to create the symbol table are _______