## Concepts Introduced in Chapter 1

- High-level introduction to compiling.
  - Phases
  - Compiler Construction Tools
  - Front Ends and Back Ends
  - Analysis-Synthesis Model
  - Assemblers
  - Linkers and Loaders

## Translator

- A translator is a program that reads a program written in a source language and translates it to an equivalent program written in a target language.



source program → translator → target program

translator → diagnostic messages
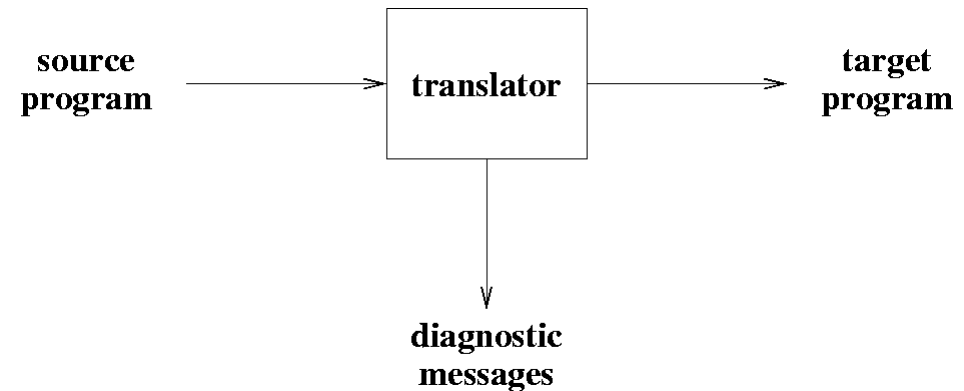
## Knowledge Required for Implementing a Successful Compiler

- Programming Languages
- Computer Architecture
- Formal Languages
- Algorithms
- Graph Theory
- Software Engineering

## Other Applications Related to Compilers

- Compiler Relatives
  - Interpreters
  - Structure Editors
  - Pretty Printers
  - Static Checkers
  - Debuggers
- Other Applications
  - Text Formatters
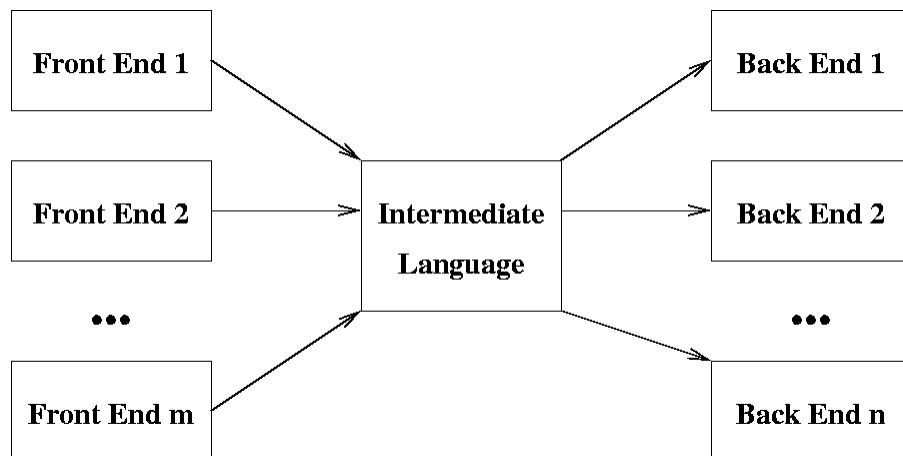  - Silicon Compilers
  - Query Interpreters

# Compiler Phases

| Phase | Output | Sample |
|---|---|---|
| programmer | source string | A=B+C; |
| scanner | token string | A, =, B, +, C, ; |
| parser | tree | (tree diagram: A = + ; with B C) |
| intermediate code generator | quads | t12 = float C<br>A = B float add t12 |
| optimizer | quads | A = B float add t9 |
| code generator | assembly code | movf C,r1<br>addf2 r1,r2<br>movf r2,A |
| peephole optimizer | assembly code | addf2 C,r2<br>movf r2,A |

# Compiler Construction Tools

- Front End (Analysis)
    - Scanner Generators: Lex
    - Parser Generators: Yacc
    - Syntax-Directed Translation Engines
- Back End (Synthesis)
    - Automatic Code Generators
    - Peephole Optimizer Construction Tools

# Front Ends and Back Ends



Front End 1, Front End 2, ... Front End m → Intermediate Language → Back End 1, Back End 2, ... Back End n

# Analysis-Synthesis Model of Compilation

- Analysis Part
    - Breaks up the source program into pieces and creates an intermediate representation.
- Synthesis Part
    - Constructs a target program from the intermediate representation.

# 3 Phases of Analysis in a Compiler

- Linear Analysis
  - Read a stream of characters and group into tokens.
- Hierarchical Analysis
  - Group tokens into hierarchical structures.
- Semantic Analysis
  - Perform certain checks to ensure that the program components fit together correctly.

# Linear Analysis

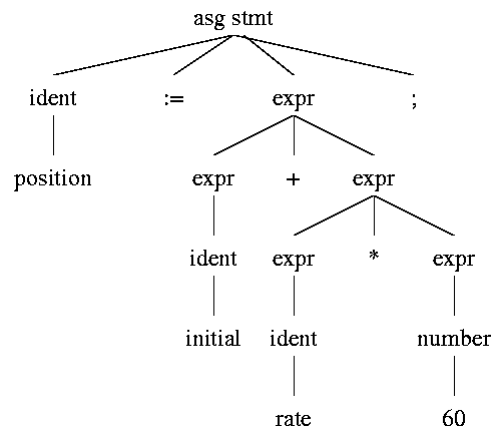- In a compiler this is also called lexical analysis or scanning.

```
    position := initial+rate*60;
=>
    position, :=, initial, +, rate, *, 60, ;
```

# Hierarchical Analysis

- In a compiler this is called parsing or syntax analysis.
- It is usually expressed in a set of recursive rules called a grammar.
- Can be represented in a parse tree.



# Semantic Analysis

- Checks for errors that can't be checked through syntax analysis alone.
  - Consistent use of types.
  - Variables declared before referenced.
- Determines where conversions need to be performed.

# Intermediate Code Generation

- After analysis, most compilers generate an intermediate representation of a program.
- Properties
  - machine-independent
  - easy to translate to the target machine language
- Can have a common intermediate language that is the target of several front ends and is input to several back ends.

# Code Optimization

- Often performed on intermediate code.
- Sometimes performed after code generation.
- Goals
  - Make a program run faster.
  - Make a program take up less space.
  - Make a program use less power.
- Should never change the semantic behavior of the program.

# Code Generation

- Produces assembly or object code from the intermediate representation.
- Each intermediate operation is translated to an equivalent sequence of machine instructions.
- Special features of the architecture are exploited.

# Preprocessors

- Perform some preliminary processing on a source module.
  - definitions and macros
    - #define
  - file inclusion
    - #include
  - conditional compilation
    - #ifdef
  - line numbering
    - #line

# Assemblers

- Typically accomplished in 2 passes.
  - Pass 1: Stores all of the identifiers representing addresses or values in a table.
  - Pass 2: Translates the instructions and data into bits for the machine code.
- Produces relocatable code.

# Linkers and Loaders

- Linker
  - Resolves external references.
  - Includes appropriate libraries.
  - Produces an executable file.
- Loader
  - Creates a process from the executable.
  - Loads the process (or a portion of it) into main memory.
  - Produces absolute machine code.