What is a Compiler?

- Tool to **translate** one language to another
  - **Source** language translated to **target** language
- Languages could be anything, but
  - Source language is usually a **high level** language
  - Target language is usually a low level **machine** language
- Examples
  - C or C++ compiles to binary machine code
  - Java compiles to Java byte code (virtual machine code)

Compiler Input and Output

- Compilers generally deal with input files
  - Source of a program – but really this is just a sequence of characters
- Output of a compiler is an executable program
  - Typically a binary file of machine code in a format that is loadable by the operating system
- Compilers may produce other artifacts of the process
  - A table of the symbols appearing in a program
  - Error messages and warnings
  - Intermediate files (relocatable object files, ASM files, etc.)
A compiler is a complex tool and is typically organized to accomplish its work in stages.

- We call these parts of the process the **phases** of the compiler.

- Each phase takes the translation/analysis further along toward the goal of the final target.

- Other compiler related tools (program analyzers, symbol lookup, debuggers, profilers) may use phases of the compiler.
A Small Example

- Begin with source code:
  \[ a[index] = 4 + 2 \]
- Scanner turns this into sequence of tokens:
  \[
  \begin{align*}
  &\text{ID} \\
  &\text{Lbracket} \\
  &\text{ID} \\
  &\text{Rbracket} \\
  &\text{AssignOp} \\
  &\text{Num} \\
  &\text{AddOp} \\
  &\text{Num}
  \end{align*}
  \]

---

Grammar specifies a complete Parse tree:

```
expression
   | assign-expression
   |   | expression = expression
   |   |   | subscript-expression
   |   |   |   | expression [ expression ]
   |   |   | expression + expression
   |   | identifier a
   |   | identifier index
   | number 4
   |   | number 2
```
Parser produces a Syntax tree - a "trimmed" version of the parse tree with only essential information:

Semantic Analysis produces a syntax tree annotated with attributes:
Code generator produces Target code

```plaintext
mov eax, 6
mov ecx, DWORD PTR _index$[ebp]
mov DWORD PTR _a$[ebp+ecx*4], eax
```

Bootstrapping and Porting

- A compiler is just a program
  - A compiler takes a program (source) as input
- Who compiles a compiler?
  - Another compiler
- Specify in terms of languages:
  - Input language to translator
  - Output language from translator
  - Implementation language of translator
The TINY Language

Syntax of a TINY program
- Sequence of statements separated by **separators**
- No procedures or declarations
- All variables are integers, implicitly declared by assigning to them with `:=`
- Control statements are **if-then-else-end** (the else is optional) and **repeat-until**
- **read** and **write** statements do I/O
- Comments are contained in curly braces (no nesting)
- Expressions are Boolean (`<`, `=`) and integer arithmetic (`+`, `-`, `*`, `/`)
A TINY Program

{ Sample program
  in TINY language –
  computes factorial
}
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
write fact { output factorial of x }
end

Execution Environment

- Tiny Machine simulator implemented by a C program
- Simple instruction set
- Source listing in Appendix C of book
- Instructions, operation described in Chapter 8
- Don't worry about it till we get to code generation