Introduction to Compiler Construction

ASU Textbook Chapter 1

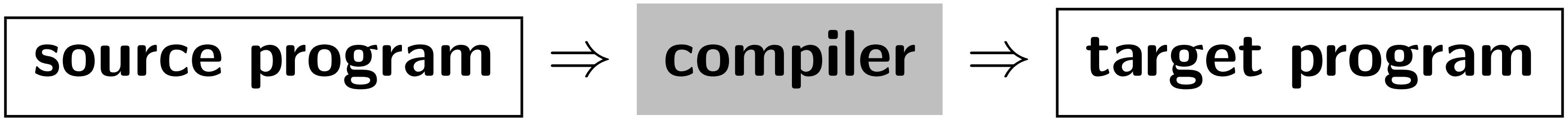
Tsan-sheng Hsu

tshsu@iis.sinica.edu.tw

http://www.iis.sinica.edu.tw/~tshsu

What is a compiler?

- **Definitions:**
  - A recognizer.
  - A translator.



source program ⇒ compiler ⇒ target program

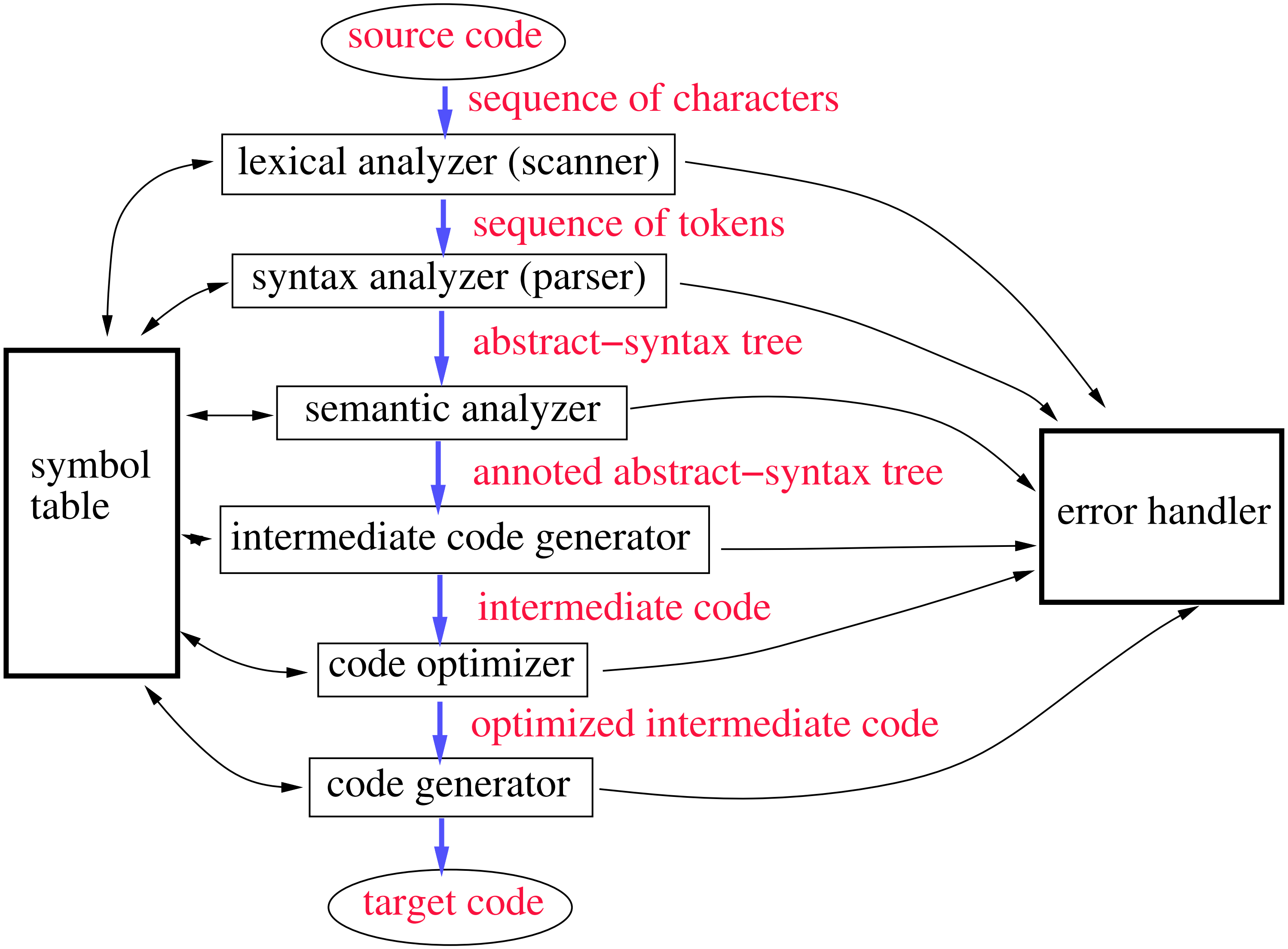
  - Source and target must be equivalent!
- **Compiler writing spans:**
  - programming languages
  - machine architecture
  - language theory
  - algorithms and data structures
  - software engineering
- **History:**
  - 1950: the first FORTRAN compiler took 18 man-years;
  - now: using software tools, can be done in a few months as a student's project.

Applications

- **Computer language compilers.**
- **Translator: from one format to another.**
    - **query interpreter**
    - **text formatter**
    - **silicon compiler**
    - **infix notation → postfix notation:**

      ```
      3 + 5 - 6 * 6          ====>          3  5  + 6  6 * -
      ```
    - **pretty printers**
    - · · ·
- **Computational theory:**
    - **power of certain machines**
      ≡ the set of languages that can be recognized by this machine;
    - **grammar ≡ definition of this machine.**

Flow chart of a typical compiler



Scanner

- **Actions:**
  - **Reads characters from the source program;**
  - **Groups characters into LEXEMES (sequences of characters that "go together") following a given pattern;**
  - **Each lexeme corresponds to a TOKEN**
    - ▷ *the scanner returns the next token (plus maybe some additional information) to the parser;*
  - **The scanner may also discover lexical errors (i.e., erroneous characters).**
- **The definitions of what a lexeme, token or bad character is depend on the definition of the source language.**

Scanner example for C

- **Lexeme: C sentence**

L1: x = y2 + 12;

(Lexeme)	L1	:	x	=	y2	+	12	;
(Token)	ID	COLON	ID	ASSIGN	ID	PLUS	INT	SEMI-COL

- **Arbitrary number of blanks between lexemes.**
- **Erroneous sequence of characters (not parts of comments) for C language:**
  - control characters
  - @
  - 2abc

Parser

- **Actions:**
  - **Group tokens into grammatical phrases, to discover the underlying structure of the source**
  - **Find syntax errors, e.g., the following C source line:**

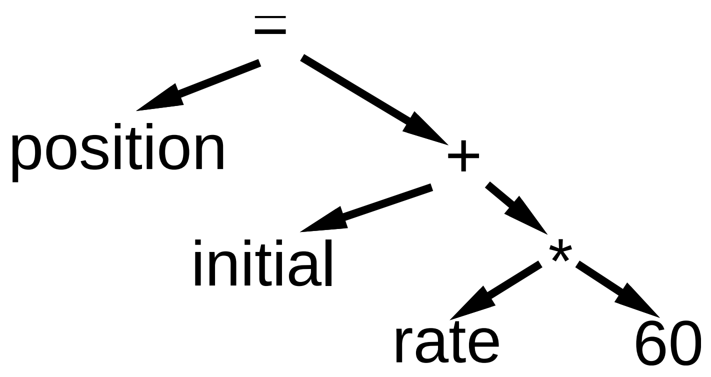
(Lexeme)	**index**	**=**	*****	**12**	**;**
(Token)	**ID**	**ASSIGN**	**TIMES**	**INT**	**SEMI-COL**

Every token is legal, but the sequence is erroneous!

- **May find some static semantic errors, e.g., use of undeclared variables or multiple declared variables.**
- **May generate code, or build some intermediate representation of the source program, such as an abstract-syntax tree.**

Parser example for C

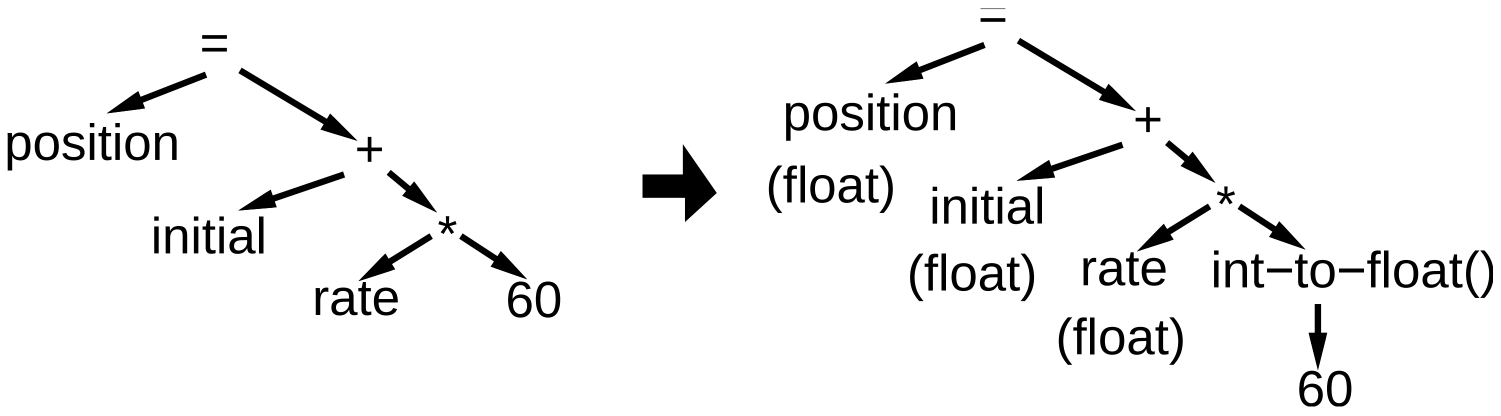
- **Source code:** **Position = initial + rate * 60;**
- **Abstract-syntax tree:**



- **interior nodes of the tree are OPERATORS;**
- **a node's children are its OPERANDS;**
- **each subtree forms a logical unit.**
- **the subtree with * at its root shows that multiplication has higher precedence than +, this operation must be performed as a unit, not "initial + rate".**

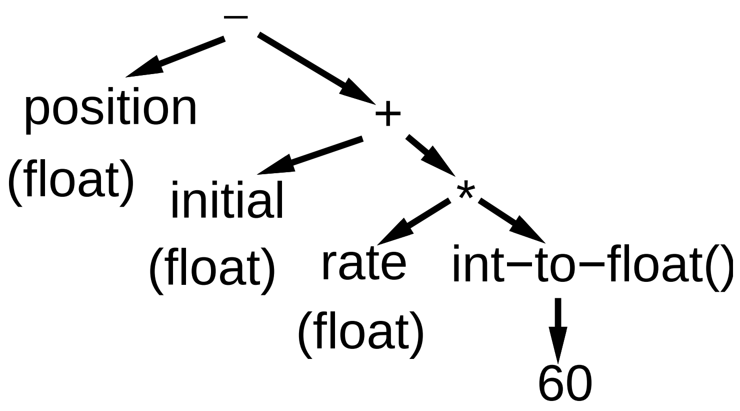
Semantic analyzer

- **Actions:**
  - **Check for more static semantic errors, e.g., type errors.**
  - **May annotate and/or change the abstract syntax tree.**



Intermediate code generator

- **Actions: translate from abstract-syntax tree to intermediate code.**
- **One choice for intermediate code is 3-address code : Each statement contains**
  - **at most 3 operands;**
  - **in addition to ":=" (assignment), at most one operator;**
  - **an"easy" and "universal" format to be translated into most assembly languages.**
- **Example:**



```
temp1 := int-to-float(60)

temp2 := rate * temp1

temp3 := initial + temp2

position := temp3
```

Optimizer

- Improve the efficiency of intermediate code.
- Goal may be to make code run faster, and/or make the code smaller and/or using least number of registers and/or less power consumption . . .
- Example:

```
temp1 := int-to-float(60)

temp2 := rate * temp1

temp3 := initial + temp2

position := temp3
```

⟹

```
temp2 := rate * 60.0

position := initial + temp2
```

- Current trend: to obtain smaller, but maybe slower, equivalent code for embedded systems.

Code generation

- **A compiler may generate**
  - **pure machine codes (machine dependent assembly language) directly, which is rare now ;**
  - **virtual machine code.**
- **Example:**
  - **PASCAL → compiler → P-code → interpreter. → execution**
  - **Speed is roughly 4 times slower than running directly generated machine codes.**
- **Advantages:**
  - **simplify the job of a compiler;**
  - **decrease the size of the generated code: 1/3 for P-code ;**
  - **can be run easily on a variety of platforms**
    - *P-machine is an ideal general machine whose interpreter can be written easily;*
    - *divide and conquer;*
    - *recent example: JAVA.*

Code generation example

```
temp2 := rate * 60.0
position := initial + temp2
```

$\Longrightarrow$

LOADF rate, R_1
MULF #60.0, R_1
LOADF initial, R_2
ADDF R_2, R_1
STOREF R_1, position

Practical considerations

- **Preprocessing phase:**
  - **macro substitution:**
    - ▷ *#define MAXC 10*
  - **rational preprocessing: add new features for old languages.**
    - ▷ *BASIC*
    - ▷ *C*
  - **compiler directives:**
    - ▷ *#include <stdio.h>*
  - **non-standard language extensions.**

Practical considerations II

- **Passes of compiling**
  - **First pass reads the text file once.**
  - **May need to read the text one more time for any forward addressed objects, i.e., anything that is used before its declaration.**
  - **Example: C language**

```
goto error_handling;
...
error_handling:
...
```

Reduce number of passes

- Each pass takes I/O time.
- **Back-patching**: leave a blank slot for missing information, and fill in the empty slot when the information becomes available.
- Example: C language
  when a label is used
  - if it is not defined before, save a trace into the to-be-processed table
    - ▷ *label_name corresponds to LABEL_TABLE[i]*
  - code generated: GOTO LABEL_TABLE[i]

  when a label is defined
  - check known labels for redefined labels
  - if it is not used before, save a trace into the to-be-processed table
  - if it is used before, then find its trace and fill the current address into the trace
- **Time and Space trade-off!**