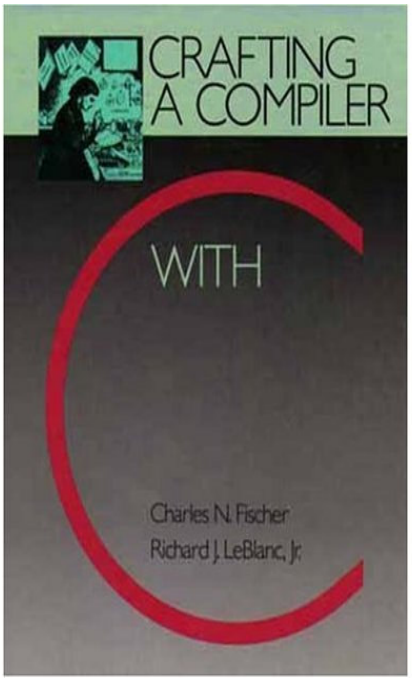
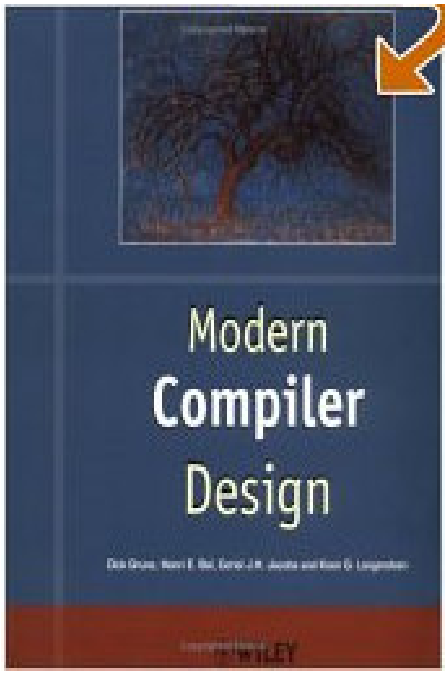
Crafting a Compiler with C (I)

資科系
林偉川





Course OUTLINE

- Modern Compiler Design, Dick Grune, 2000
- Crafting a Compiler with C, Benjamin/Cummings,1991
- Midterm ➔30%
- Final exam ➔30%
- Attend the class ➔10%
- 30%
  - Home Work
  - Parser program by compiler tools such as Lex、Yacc

3

Pre-requirement Background

- Programming Language in C or C++ or Java
- Programming Language Structure (The concept of Programming Language)
- Automata Theory

4

Why study compiler

- Apply the technologies such as hashing, stack, garbage collection, dynamic programming, tree and graph algorithms
- Using a parser generator, a parser can be generated automatically. Especially for file conversion such as word ➔ PDF conversion
- Can design for the command processing such as Linux C-shell

5

Introduction

- Compilers allow programs and programming expert to be machine-independent

6

First real compiler

- FORTRAN compiler took 18 person-years to build
- Today compilation techniques are well understood, and a simple compiler can be built in a few months
- Crafting an efficient and reliable compiler is still a challenging task

7

Compiler Theory Application

- Translating text and formatting command
  - Latex or PDF maker
  - UNIX command
    - nroff and troff translating text and formatting command
    - Eqn for handling equation (Equation Editor)
    - Tbl for formatting table (insert table by rows/columns)
    - Pic for drawing pictures (insert picture by existed picture)
- Hardware Layout simulation
  - Silicon compiler
    - ORCAD or ICE

8

Compiler Classification

Compilers are distinguished according to the generated target code

- Pure machine code (Non-OS)
  - Compilers generate code for a particular machine's instruction set, not assuming the existence of any OS or library routine
  - This form of target code can be executed on bare HW without dependence on any other SW

9

Compiler Classification

- Augmented machine code
  - Compilers generate machine code for a machine architecture augmented with OS and language support routines such as I/O, storage allocation, mathematical functions
- Virtual machine code (Java)
  - The generated code is composed of virtual instructions for producing a transportable compiler
  - A simulator for the virtual machine is needed by the compiler (P-code for Pascal, .class for Java)

10

Micro-program

- Some computers can be changed via microprogramming which is an ideal mechanism for interpreting a virtual instruction set. [by re-flash-ROM]

➔ Almost all compilers generate code for a virtual machine, some of those operations must be interpreted in SW or firmware

11

Target machine code format

Target machine code formats can be classified:

- Assembly Language format
  - Generated assembler code to simplify translation
  - For cross compilation ➔ running a compiler on one computer with its target language being the machine language of a second computer [transfer easily]
  - Easy to check the correctness of a compiler by observing the output
  - However, Assemblers are slow and machine-dependent ➔pseudo Assembly

12

Target machine code format

- Re-locatable binary format
  - Target code is generated in a binary format in which external references, local instruction and data addresses are not yet bound
  - Addresses are assigned relative to the beginning of the module or symbolic named location
  - A linkage step is required to add any support libraries and other precompiled routine to produce an absolute binary program format ➔executable

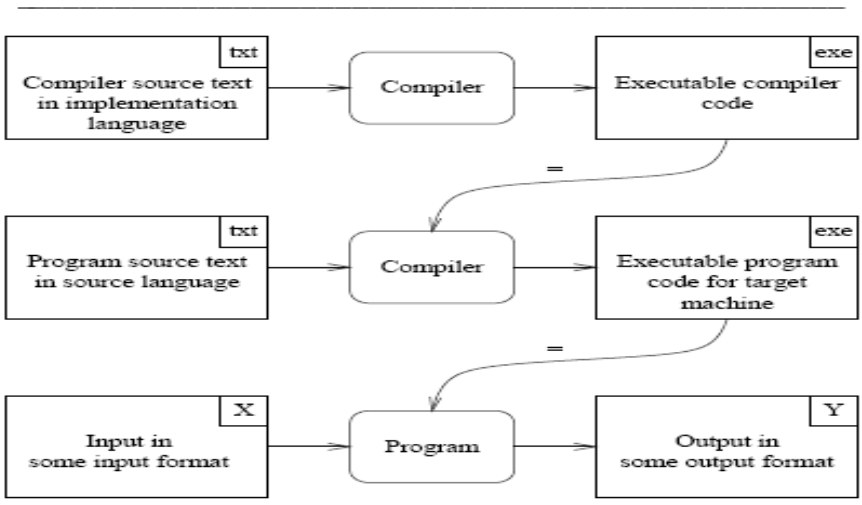
13

Target machine code format

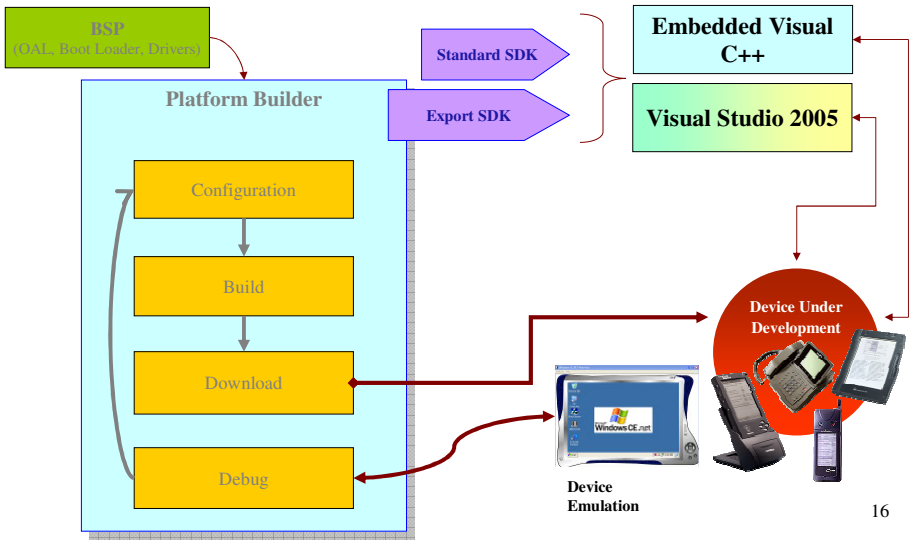
- Memory-image (Load-and-Go) format
  - The compiled output may be loaded into the compiler's space and executed immediately
  - No linkage step, so allow a program to be prepared and executed in one step
  - The ability to interface with external, library, and precompiled routines is limited
  - The program must be recompiled for each execution unless to store the memory-image
  - Useful for student and debugging use

14

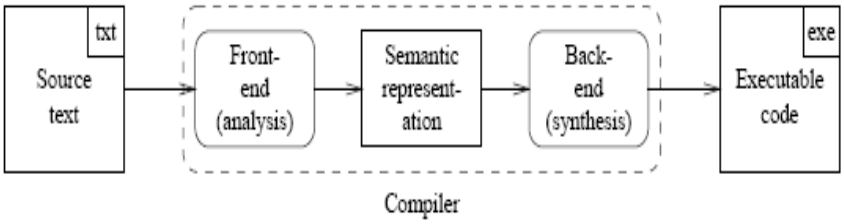
Compiling and running



系統開發流程



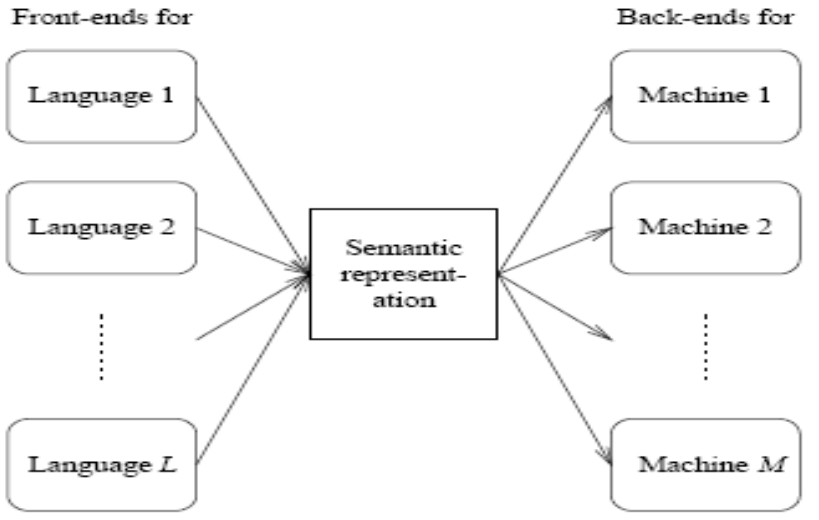
Conceptual structure of a compiler



- Front-end is un-aware of the target language and the back-end is unaware of the source language
- An interpreter is written in a high-level language and will run on most machine types, generated object code will run on machines of the target type ➔ portability is increased

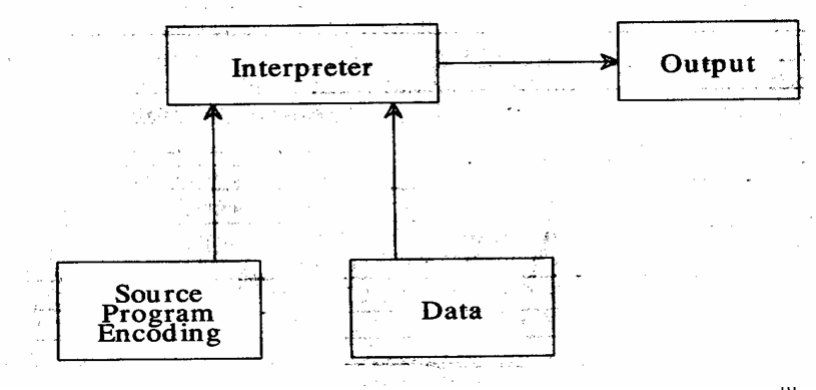
17

Creating compilers for *L* languages and *M* machines



Interpreter

- Compilers are expected to generate efficient code
- It executes programs without explicitly performing a translation



19

Interpreter advantage

- Modification or addition to user programs as execution proceeds ➔ interactive debugging
- A variable denotes the type of object may change dynamically ➔ symbols need not have a fixed meaning make direct translation into machine code impossible
- The increased security that can be achieved by interpreter (Java), compiled code could not do the same checks but an interpreter can

20

Interpreter advantage

- Better diagnostics➔ source text analysis is intermixed with execution of the program
- A significant degree of machine independent since no machine code is generated
- An interpreter does not play dirty tricks than that there are booby traps hidden in binary executable code (buffer overflow for virus!!)

21

Interpreter disadvantage

- Execution speed is slower than compiler ➔ program text must be continuously reexamined with identifier binding
- Substantial space overhead may be involved ➔ the interpreter and all support routines must be kept available. So the size penalty may lead to restrictions in the size of programs, the number of variables or procedures

22