Compilation and interpretation

Outline

- Problem: how to make a program in a high level language to execute on a computer?
  - Compilation, Interpretation, and their comparison
- Compilation
  - Details of compilation
  - Implementation of compilation

What do we teaching

- We do not teach individual programming languages here
- We talk about the common features, or concepts, of programming languages

How to make a program P in language X executable on computers

- Compilation
  - The program P in language X is translated to a program P’ in language Y by a *compiler for X.*
  - When Y is a machine language, P’ executable on a computer
  - Example: compiling a C program
- Interpretation
  - An *interpreter for X* understands this language X and can execute the program directly.
  - Example: a Unix shell / dos command shell

Compilation vs interpretation

- From designers' view

	Compiler	Interpreter
Efficiency of target program	Faster	Slower
Some features of source languages	Difficult	Easy to implement
……	…….	……

- From programmers' view
  - Developing a program is a sequence of changes
    - Each change can be easily tested on an interpreter
    - More complex in the case of compiler
  - For final product
    - The product has to be shipped with an Interpreter. The execution may be slow.
    - For compilation based, just the binary executable program shipped. The execution is normally more efficient than interpretation based.

- A trend to execute a program is to combine compilation and interpretation
  - Compile P in language X into P’ in Y, and execute P’ on an interpreter of Y. The shipped products: P’ + interpreter.
  - Why this way? One reason is from *portability* of the language. To write a compiler for X for each brand computer is much harder than a compiler for Y which is a easier language than X.
  - Example Java.
    - In more recent Java implementation, JIT (just in time) compiler is used to translate Bytecode into machine code immediate before the program in bytecode is executed, and the program in machine code will be executed (and thus much faster than Java virtual machine / bytecode interpreter).

Concepts of programming languages

- How to define a language precisely – Syntax (good for both users and designers)
- Meaning of a language (semantic analysis)
- Names (their uses and meaning) – special in programming language: manage the complexity (of a program in this language)
- Type system

Compilation

- More details about a language from a designer's view
  - Usually a program is organized into several *files.*
  - A language usually provides *libraries* which is normally not a direct part of the language but can be used by programmers (e.g., printf in C)
- Produce the executable from P in language X
  - Use a compiler to translate each file into a target file in language Y
  - Use a *linker* to combine all targeted files and libraries into the executable program.

- Major components of compiler
  - Parsing the *input program* to obtain a *parse tree*
  - *Semantic analysis* and producing *intermediate program P1*, based on the parse tree
  - Machine independent *improvement* of P1
  - From P1 generate the *target program P2*
  - Machine dependant *improvement of P2* → *P'* in the machine language

Implementation of a compiler

- One implementation strategy (bootstrapping) as an example (Pascal) is to have the following programs available
  - A compiler in Pascal: translate a program in Pascal to one in P-code
  - The same compiler in P-code
  - A P-code interpreter in Pascal
  - Given a new machine with a machine language M, what's the best way to produce a Pascal compiler executable on this machine?