ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

COMPILER CONSTRUCTION

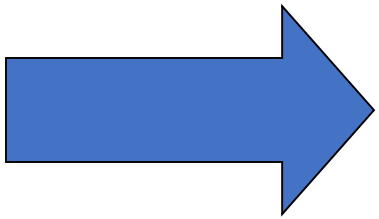
Why study compiler construction? (1)

To learn how a computer works, you cannot stop at the hardware. You need to study the operating system, file systems, **the compiler**, the network stack, ...

1. understand what is going on inside the tools that you use.
2. the theoretical and practical knowledge that is needed to implement a programming language, know the characteristics of many programming languages.
3. design and implement your own domain-specific language.
4. a gentle introduction to formal methods that are used for general purpose software design.

Why study compiler construction? (2)

5. A large variety of applications can be modelled after a compiler (or some part thereof). Simulators, debuggers, program analysis tools, editors, IDEs, RDBMSs
6. complicate techniques in text processing
7. optimization techniques
8. working with really big data structures and complex interactions between algorithms



Help you out on your next big programming project.

More details about the course

- How do computers work?

  (instruction set, registers, addressing modes, run-time data structures, ...)
- How do compilers work?
- What machine code is generated for certain language constructs?
- What is good language design?

Course Outline

- Functions of a Language Processor
- The Phases of a Compiler
- Generative Grammar
- BNF and Syntax Diagrams
- Scanner and Symbol Table
- Top Down Parsing with Backtracking
- Predictive Parsing
- LL(k) Grammars

Course Outline

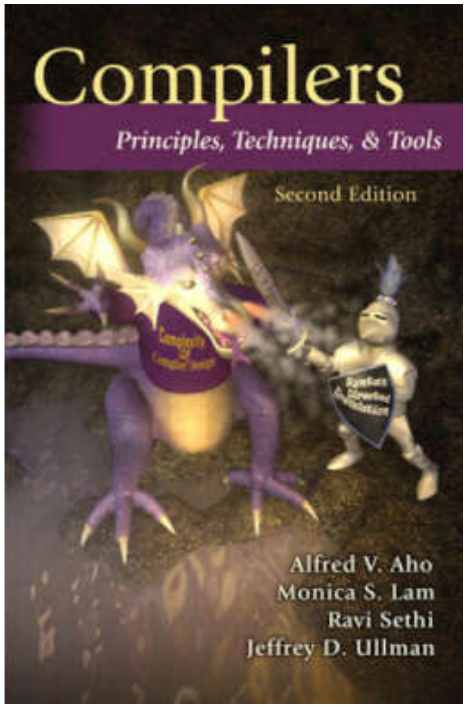
- Recursive Descent Parsing
- The Parser of KPL
- Semantic Analysis
- Stack calculator
- Intermediate Code Generation
- Object Code Generation
- Code optimization

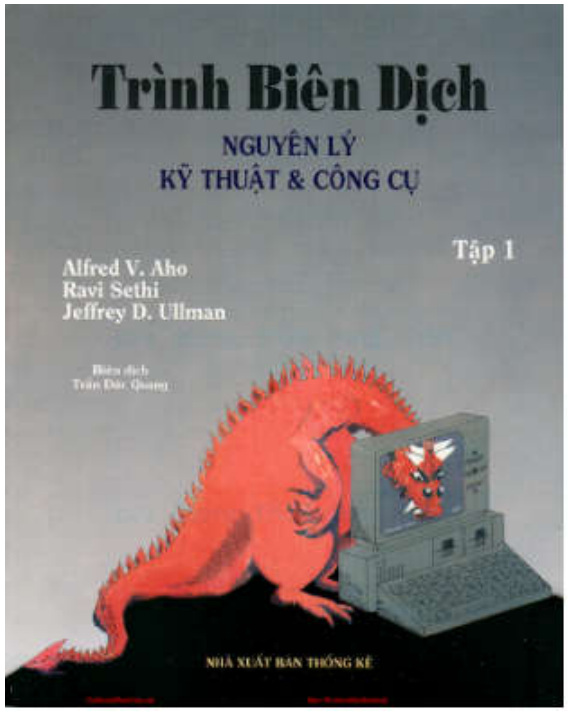
Textbooks

- Aho.A.V, Sethi.R., Ullman.J.D.
  Compiler : Principles, Techniques and Tools.
  Addison Wesley.200

Textbooks

- Aho.A.V, Sethi.R., Lam M., Ullman.J.D.
  Compiler : Principles, Techniques and Tools.
  Addison Wesley. 2007.
  Vietnamese translation (1986 edition)
  Trình biên dịch: Nguyên lý, Kỹ thuật và Công cụ
  Nhà xuất bản Thống kê, 2000





Text books

- Andrew.W.Appel
  Modern Compiler Implementation in Java
  Princeton University.1998
- Bal.H. E.
  Modern Compiler Design.
  John Wiley & Sons Inc (2000)
- William Allan Wulf.
  The Design of an Optimizing Compiler
  Elsevier Science Ltd (1980)
- Charles N. Fischer.
  Crafting a Compiler
  Benjamin-Cummings Pub Co (1987)

ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Unit 1. Functions of a Language Processor

High Level Programming Languages

- Programming languages have taken 5 generation
- A language is considered high or low level depending on its abstraction

  A high level language may use natural language elements, be easier to use, or more portable across platforms

  Low level languages are closer to the hardware

The first and the second generation

- The first generation: machine language
- The second generation : Assembly
- Languages of the first and the second generation are low level languages

The Third Generation

- Easier to read, write and maintain
- Allow declarations
- Most 3GLs supports structured programming
- Examples: Fortran, Cobol, C, C++, Basic . . . .

The Fourth Generation

- Designed with a specific purpose in mind, such as the development of commercial business software
- Reduce programming effort, cost of software development
- May include form or report builder
- Examples :SQL, Visual Basic, Oracle (SQL plus, Oracle Form, Oracle Report). . . .

The Fifth Generation

- Based around solving problems using constraints given to the program, rather than using an algorithm written by a programmer
- Are designed to make the computer solve a given problem without the programmer
- Most constraint-based and logic programming languages and some declarative languages are fifth-generation languages.

Characteristics of high-level languages

- Hardware independence
- Close to natural languages
- Easy to read, write and maintain
- Programs written in a high-level language must be translated into machine language
- Often result in slower execution speed, higher memory consumption

Syntax and Semantics of Programming Languages

- Syntax:The way symbols can be combined to create well-formed sentence (program) in the language
- Semantics: The meaning of syntactically valid strings in a language

Language Processors

- A program that performs tasks, such as translating and interpreting, required for processing a specified programming language.For example,
  - Compiler
  - Assembler
  - Interpreter
  - Compiler - Compiler

Compilers vs Interpreters

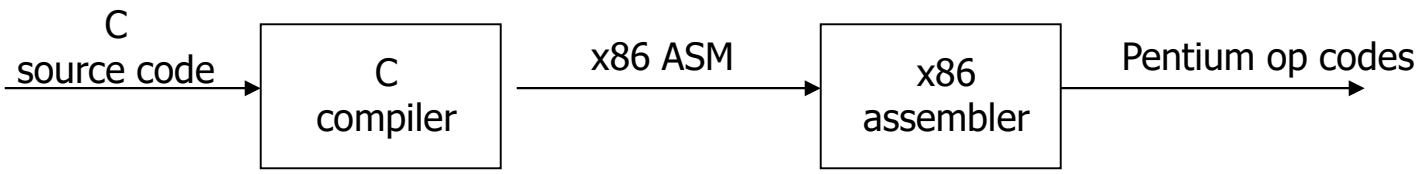
- A compiler or an interpreter is a computer program (or set of programs) that converts program written in high-level language into machine code understood by the computer

Language Translation

- Native-code compiler: produces machine code
  - Compiled languages: Pascal, C, C++, …
- Interpreter: translates into internal form and immediately executes
- Interpreted languages: Javascript, PHP, Haskell…
- Hybrid approaches: VB.net, Python, Java

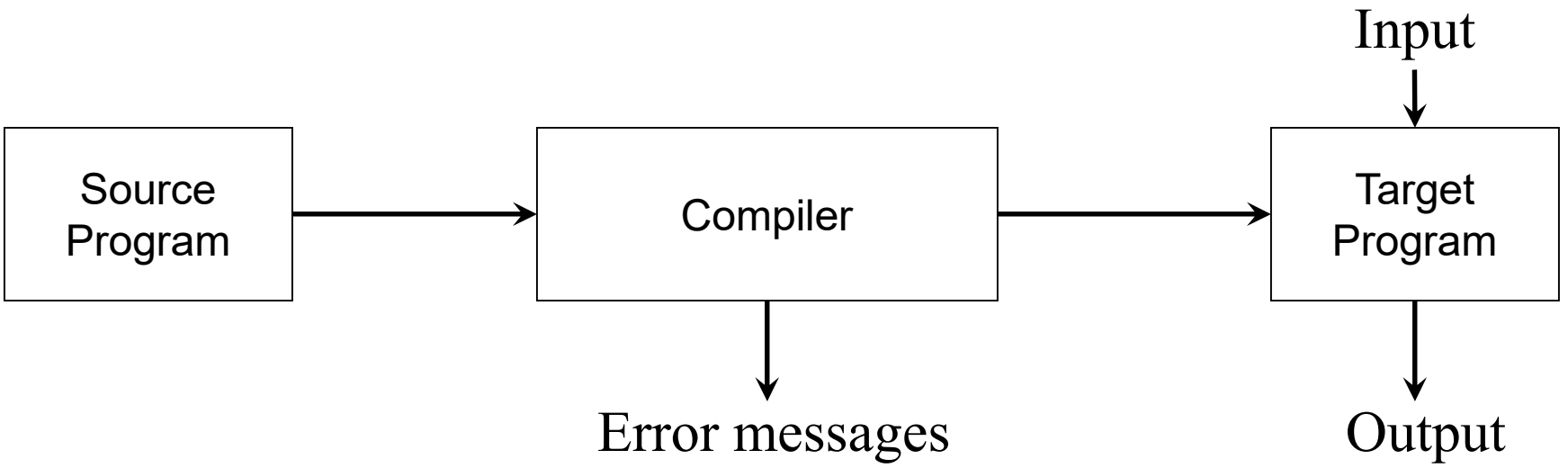
Language Compilation

- Compiler: program that translates a source language into a target language
  - Target language is often, but not always, the assembly language for a particular machine

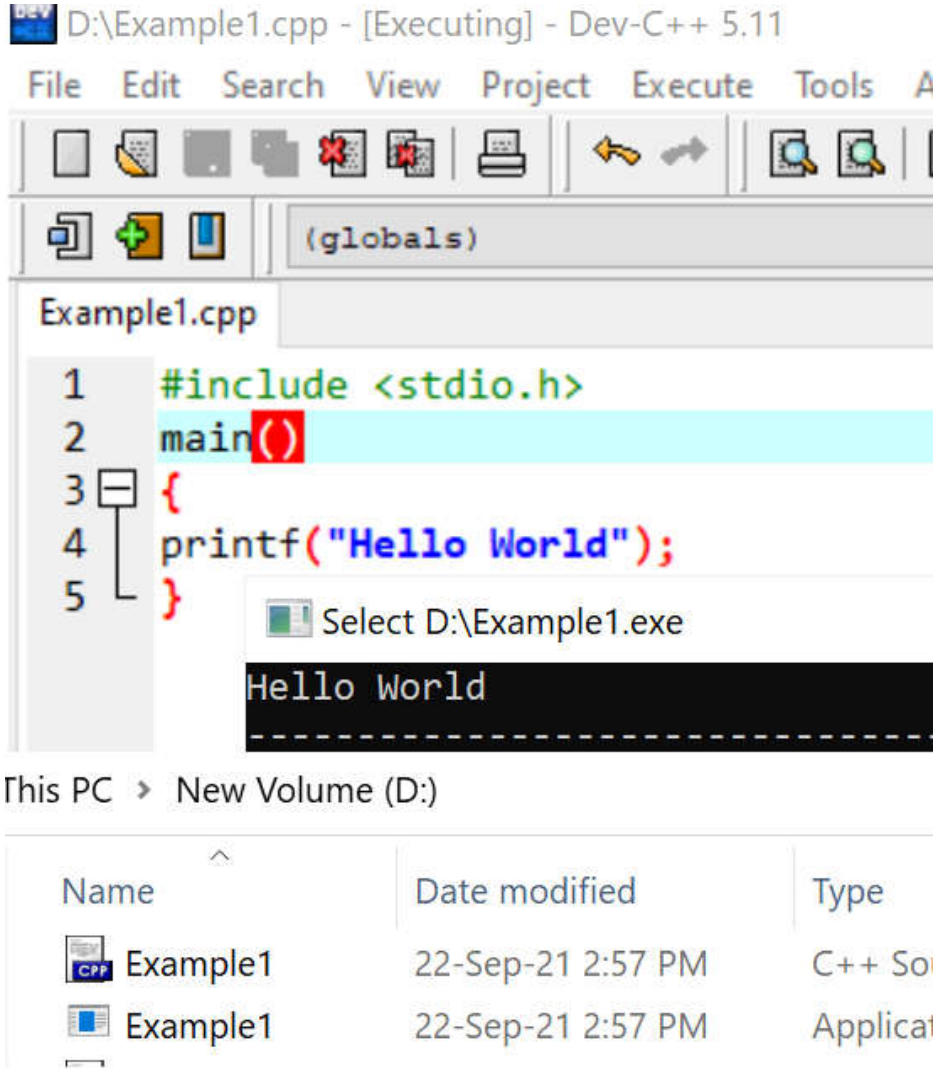


Compilers

- Scans the entire program and translates it as a whole into machine code.
- Takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
- Generates the error message only after scanning the whole program.



C translator: compiler



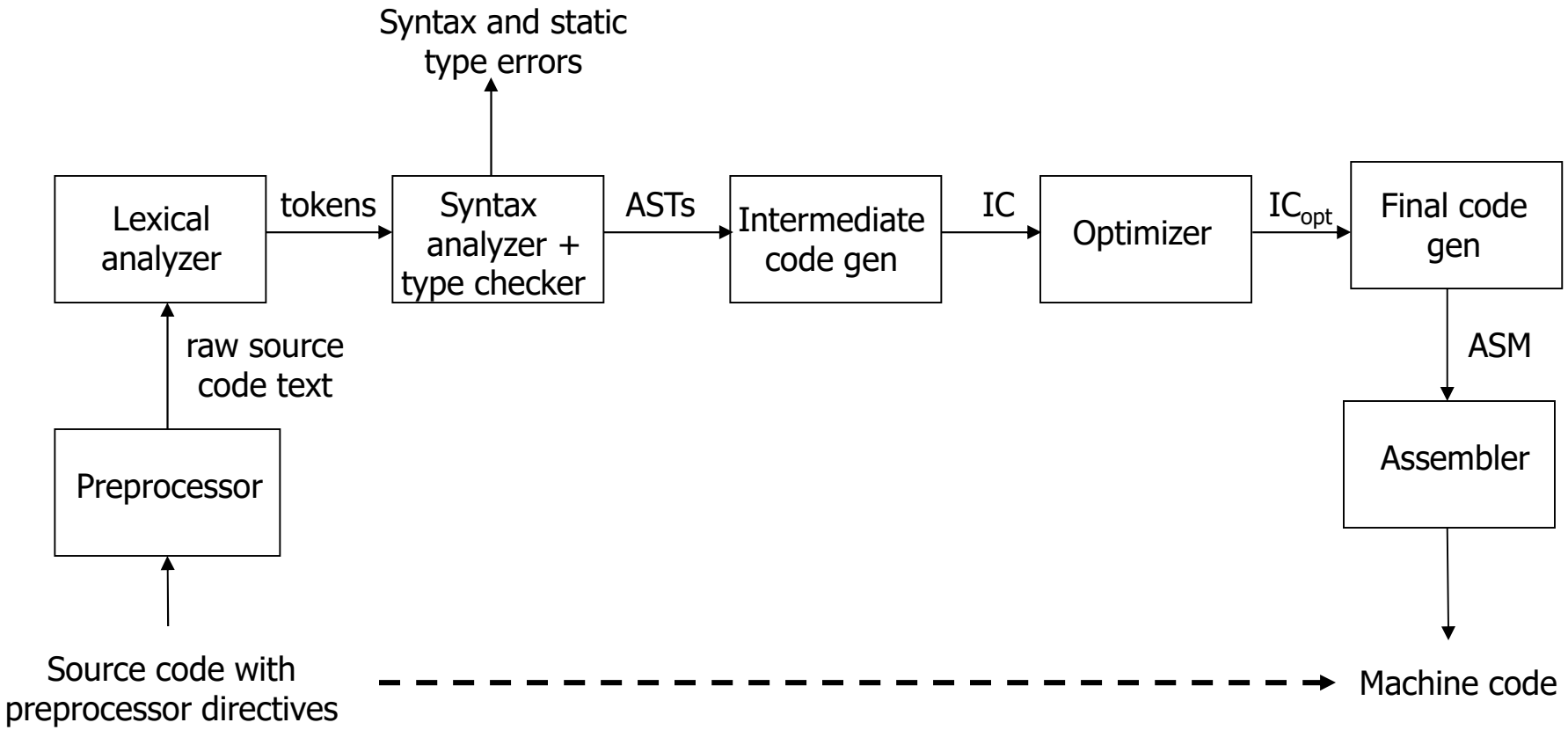
Compilers (cont'd)

- The most common reason for wanting to transform source code is to create an executable program.
- To run a program, execute the compiler (and possibly an assembler) to translate the source program into a machine language program.
- Execute the resulting machine language program, supplying appropriate input.

Checks During Compilation

- Syntactically invalid constructs
- Invalid type conversions
  - A value is used in the "wrong" context, e.g., assigning a float to an int
- Static determination of type information is also used to generate more efficient code
  - Know what kind of values will be stored in a given memory region during program execution
- Some programmer logic errors (not all compilers)
  - Can be subtle: `if (a = b) …` instead of `if (a == b) …`

Compilation Process

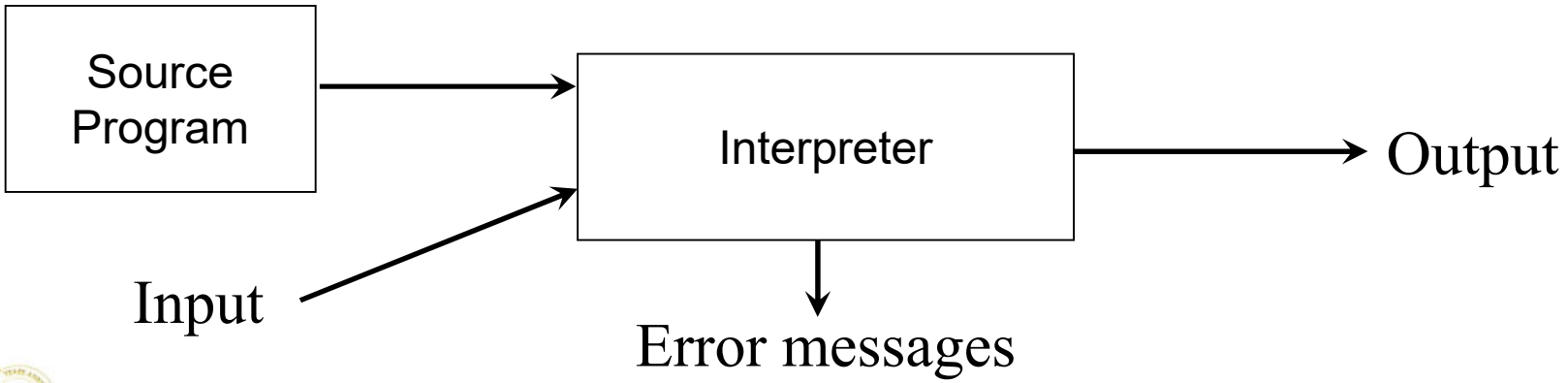


Phases of Compilation

- Preprocessing: conditional macro text substitution
- Lexical analysis: convert keywords, identifiers, constants into a sequence of tokens
- Syntactic analysis: check that token sequence is syntactically correct
- Semantic Analysis: Generate abstract syntax trees (AST), check types
- Intermediate code generation: “walk” the ASTs (or Parse Trees) and generate intermediate code
  - Apply optimizations to produce efficient code
- Final code generation: produce machine code

Interpreters

- Translates program one statement at a time
- Takes less amount of time to analyze the source code but the overall execution time is slower.
- Continues translating the program until the first error is met, in which case it stops.
- Oversimplified view:

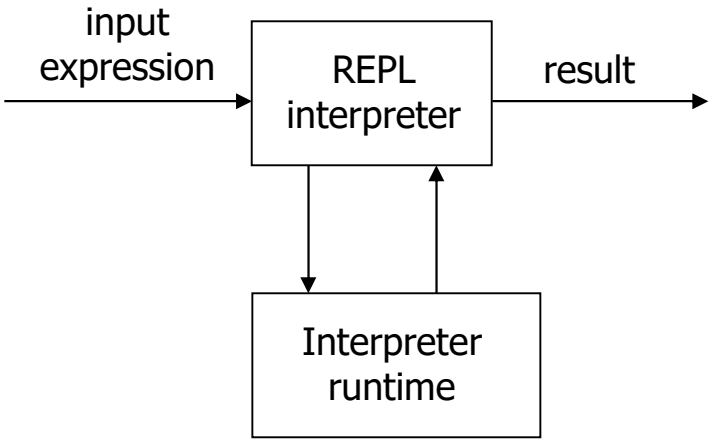


Interpreters (cont'd)

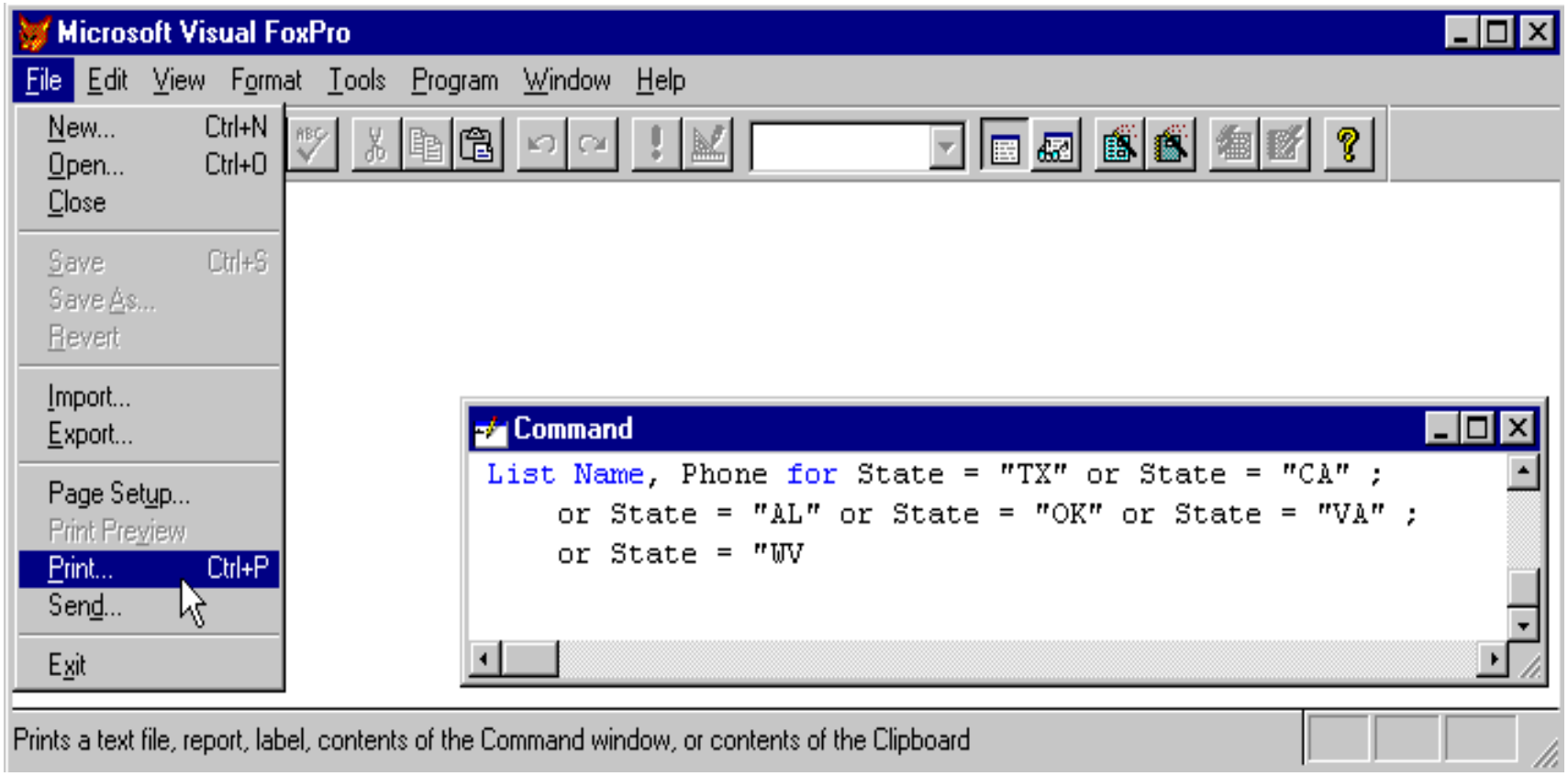
- Accepts the source language program **and** the appropriate input
- Itself produces the output of the program.

Language Interpretation

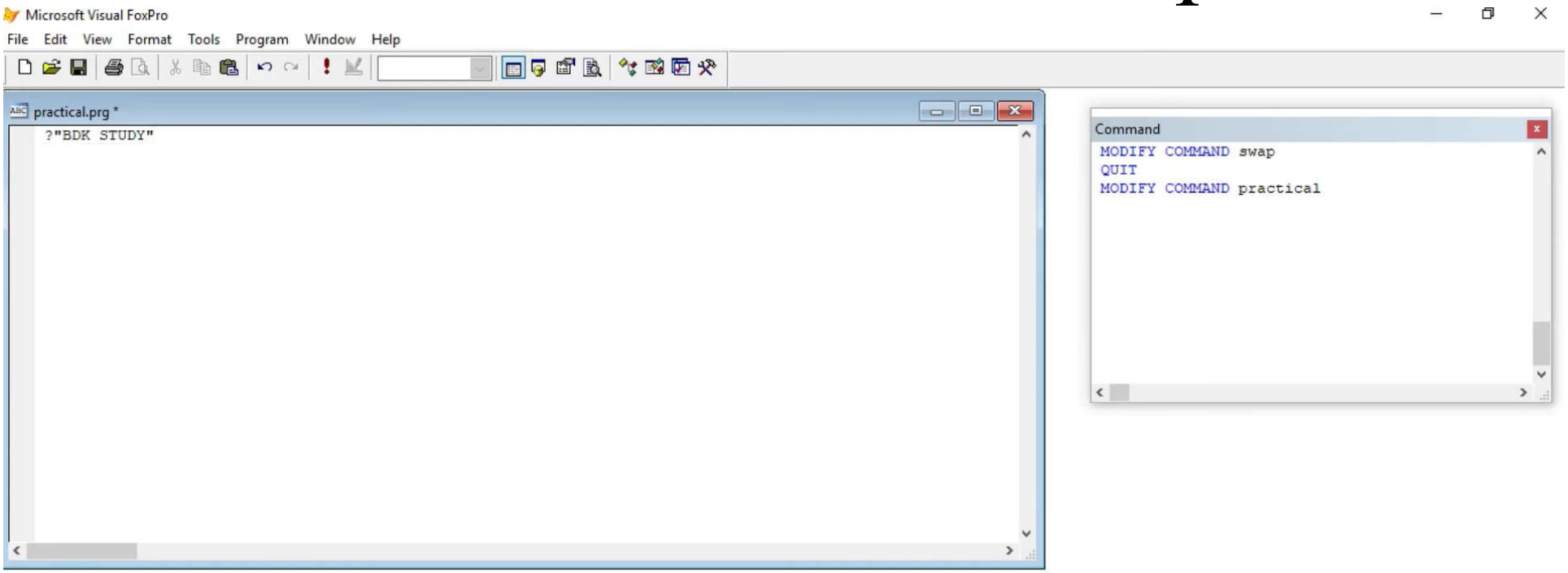
- Read-eval-print loop
  - Read in an expression, translate into internal form
  - Evaluate internal form
    - This requires an abstract machine and a “run-time” component (usually a compiled program that runs on the native machine)
  - Print the result of evaluation
  - Loop back to read the next expression



Command window in Foxpro

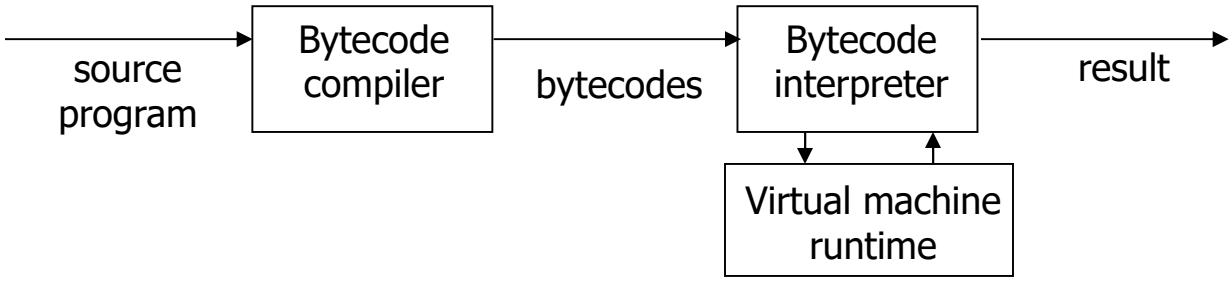


Code editor window in Foxpro



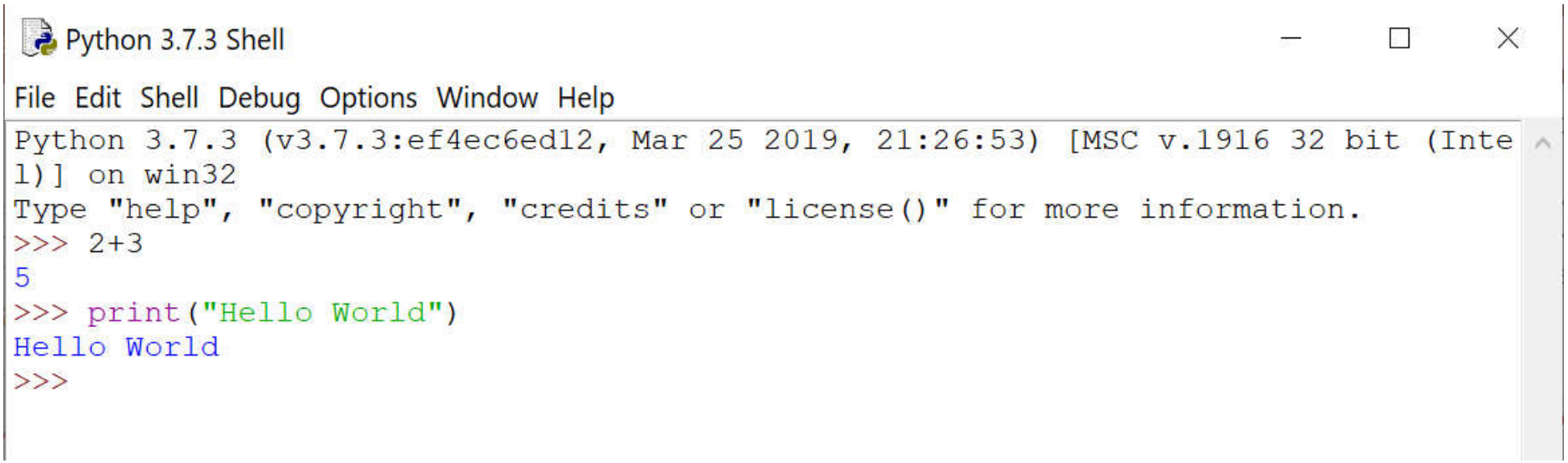
Bytecode Compilation

- Combine compilation with interpretation
  - Idea: remove inefficiencies of read-eval-print loop
- Bytecodes are conceptually similar to real machine opcodes, but they represent compiled instructions to a virtual machine instead of a real machine
  - Source code statically compiled into a set of bytecodes
  - Bytecode interpreter implements the **virtual machine**



Python translator: interpreter

Python editor window



Python translator: hybrid solution

Pros and cons of compiled and interpreted languages

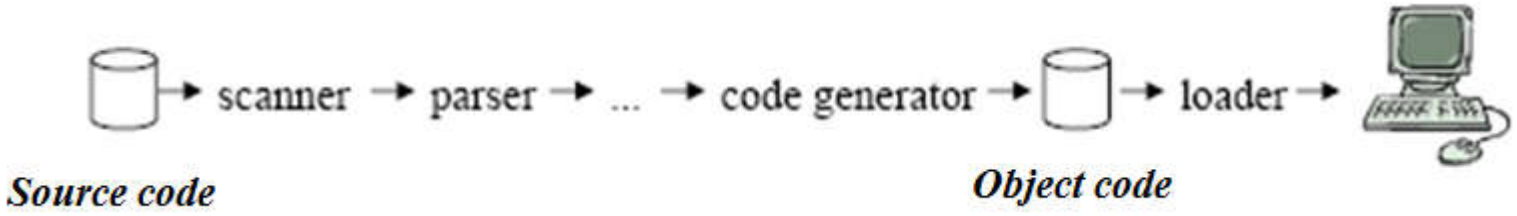
COMPILED		INTERPRETED	
ready to run	**not** cross-platform	cross-platform	interpreter required
often **faster**	inflexible	simpler to test	often **slower**
source code is **private**	extra step	easier to debug	source code is **public**

Interpreter as a part of compiler

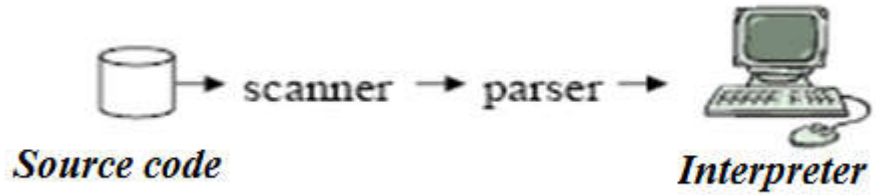
- In a compiler implementations
- The source code is compiled to a machine language for an idealized virtual machine
- The interpreter of accepts the codes and the input, produces the output.
- This technique is quite popular to make the compiler independent with the computer (portability of the compiler)

Compilers and Interpreters (cont'd)

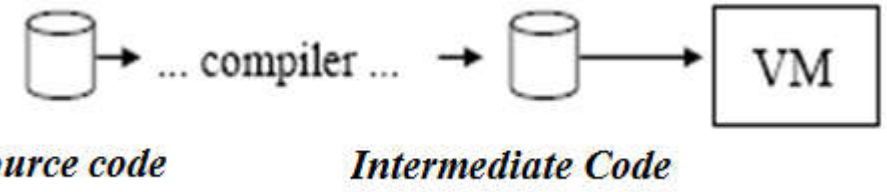
- A **Compiler** is a *program* that translates code of a *programming language* into *machine code (assembly)*



- An **interpreter** translates some form of source code into a target representation that it can immediately execute and evaluate



- **Modification of Interpreter** :a *program* that implements or simulates a *virtual machine* using the base set of instructions of a *programming language* as its *machine language*



Cousins of the compiler

- Interpreter
- Assembler
- Linker
- Loader
- Preprocessor
- Editor
- Debugger
- Profiler

The context of a compiler in a language processor