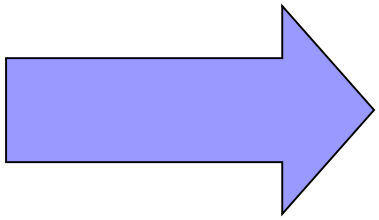
COMPILER CONSTRUCTION

Nguyen Thi Thu Huong
Department of Computer Science-HUST
Email: huongnt@soict.hust.edu.vn
Cell phone 0903253796

Why study compilers?

- Be able to build a compiler for a (simplified) (programming) language
- Be familiar with assembly code and virtual machines
- Be able to define LL(1), LL(k) grammars
- Be familiar with compiler analysis and optimization techniques
- Working with really big data structures
- Complex interactions between algorithms
- Experience with large-scale applications development



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.1986
- 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)

Text books

- Niklaus Wirth
  Compiler Construction.
  Addison Westley. 1996
- Andrew.W.Appel
  Modern Compiler Implementation in Java
  Princeton University.1998
- Nguyễn Văn Ba
  Techniques of Compiling (in Vietnamese)
  Hanoi University of Technology1994
- Vũ Lục
  Parsing(in Vietnamese)
  Hanoi University of Technology1994
- www.sourceforge.net

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

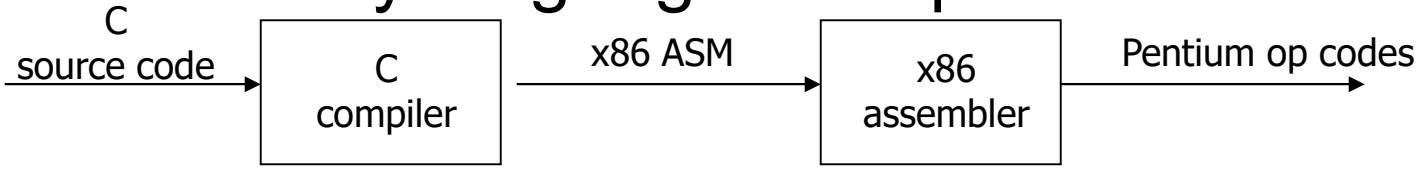
- A program that performs tasks, such as translating and interpreting, required for processing a specified programming language.For example,
  - Compiler
  - Assembler
  - Interpreter
  - Compiler - Compiler

Language Translation

- Native-code compiler: produces machine code
  - Compiled languages: Fortran, C, C++, SML …
- Interpreter: translates into internal form and immediately executes (read-eval-print loop)
  - Interpreted languages: Scheme, Haskell, Python …
- Byte-code compiler: produces portable bytecode, which is executed on virtual machine (e.g., Java)

Language Compilation

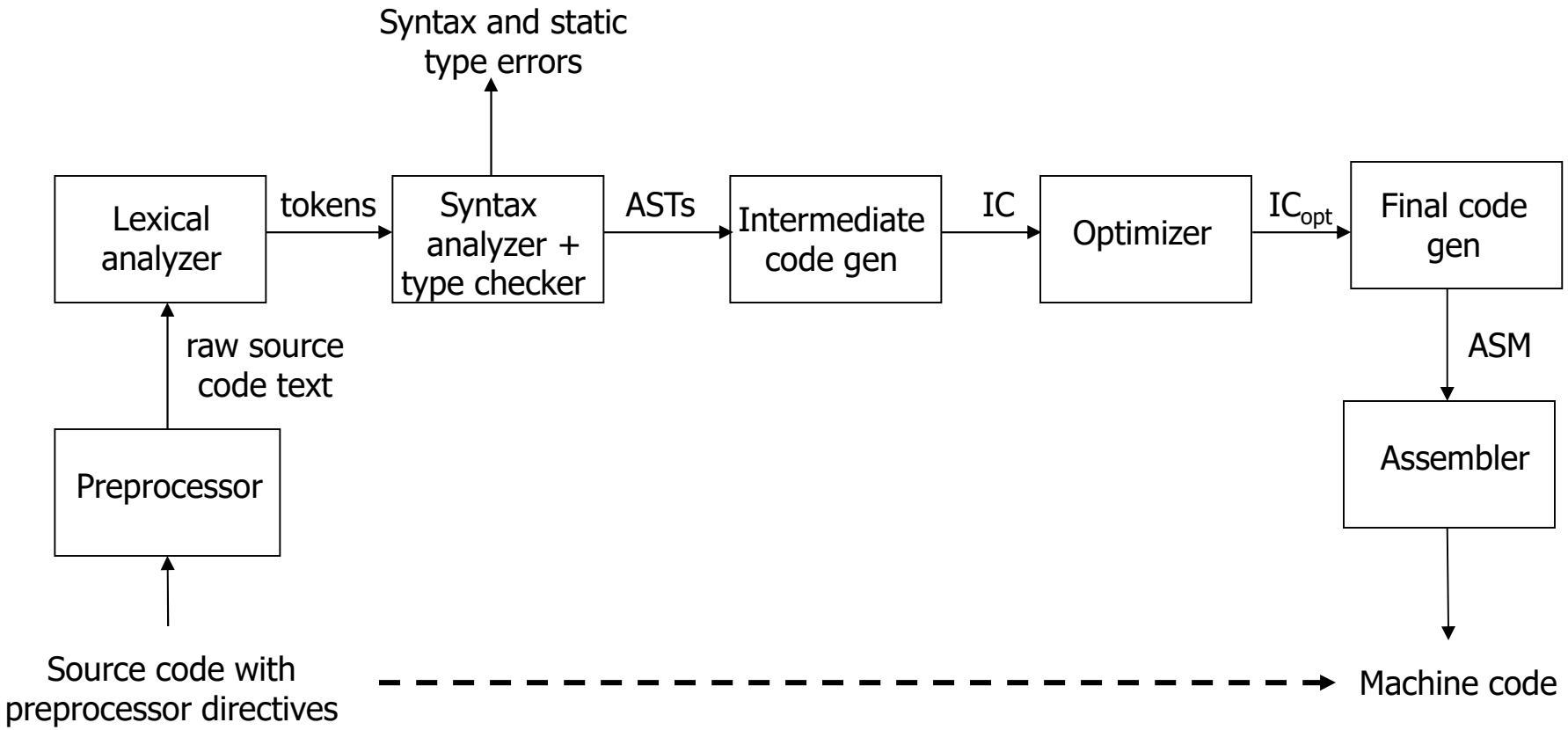
- Compiler: program that translates a source language into a target language
  - Target language is often, but not always, the assembly language for a particular machine



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

Compilation Process

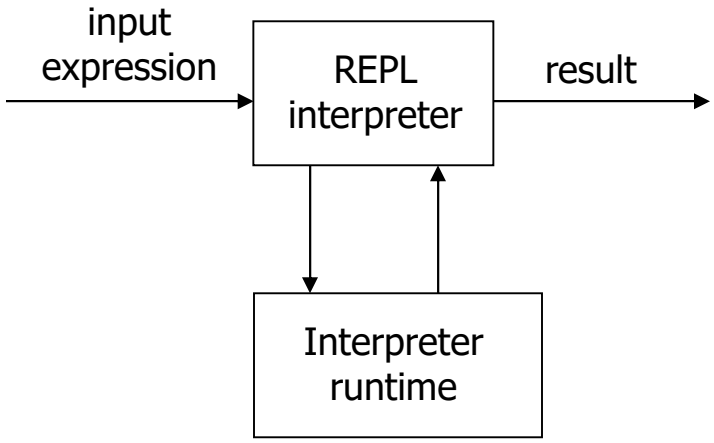


Phases of Compilation

- 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
  - 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

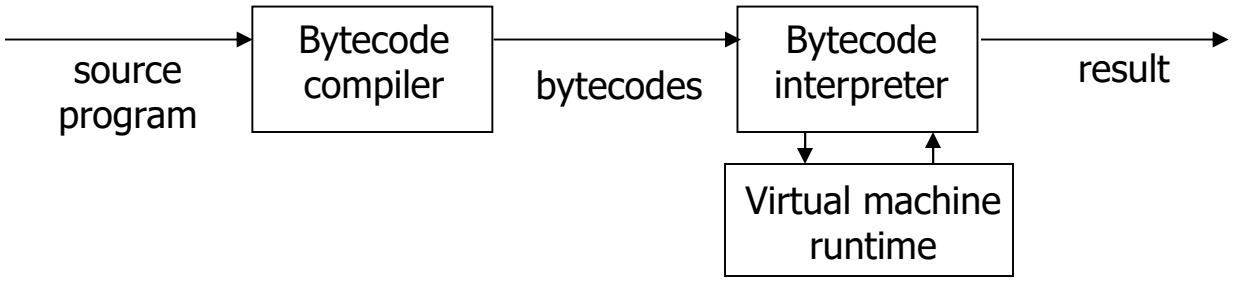
Language Interpretation

- 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



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



Binding

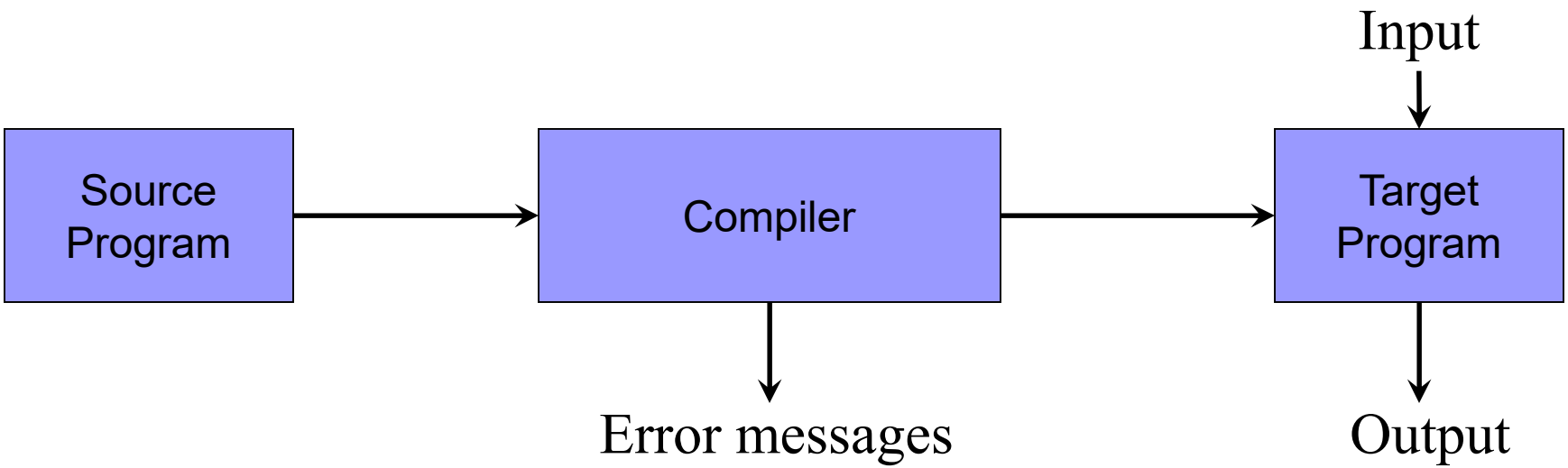
- Binding = association between an object and a property of that object
  - Example: a variable and its type
  - Example: a variable and its value
- A language element is bound to a property at the time that property is defined for it
  - Early binding takes place at compile-time
  - Late binding takes place at run-time

Compilers vs Interpreters

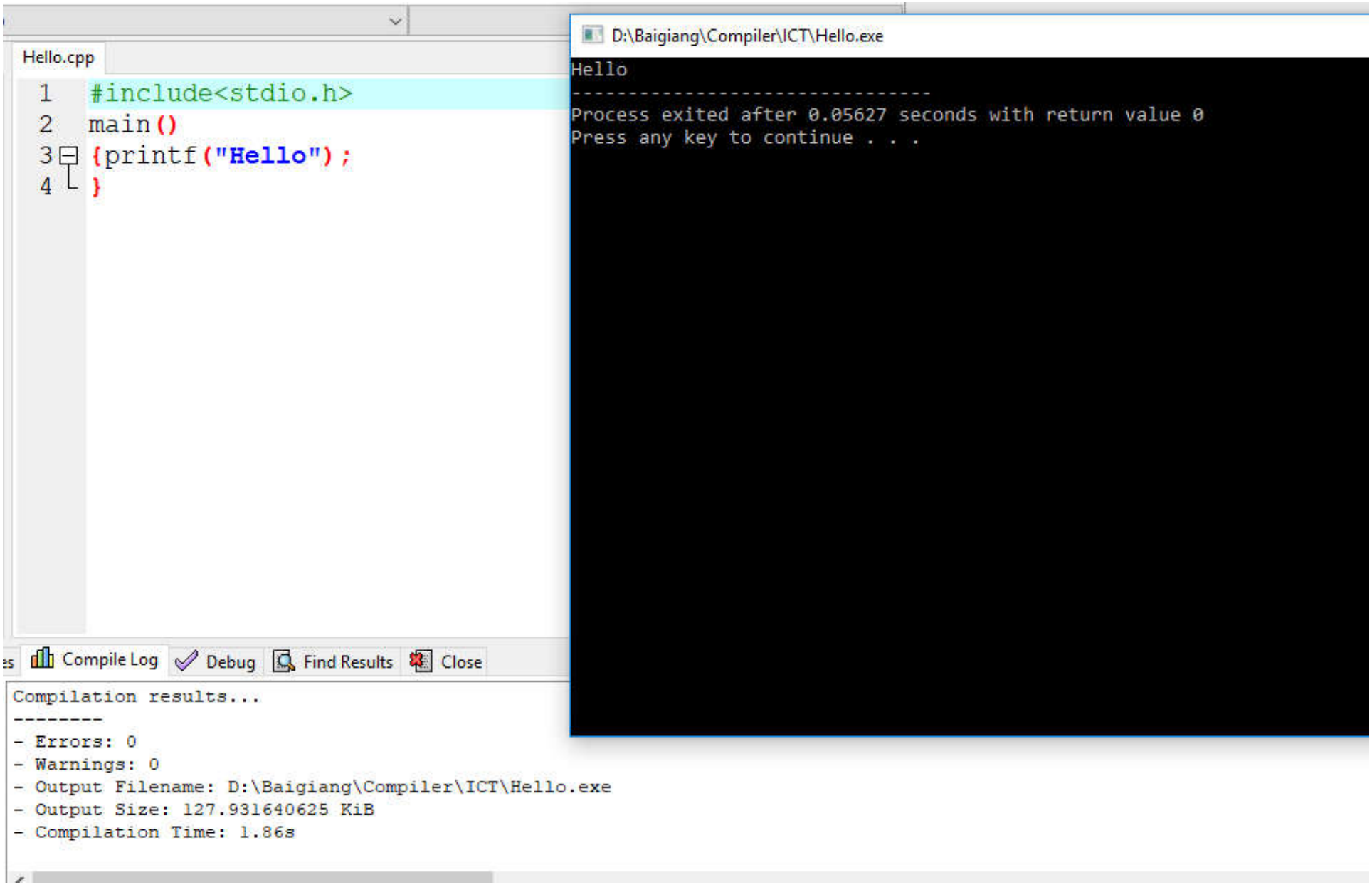
- A compiler or an interpreter is a computer program (or set of programs) that converts program written in high-level language into machine code understood by the computer

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.
- Programming language like C, C++ use compilers.



C translator: compiler

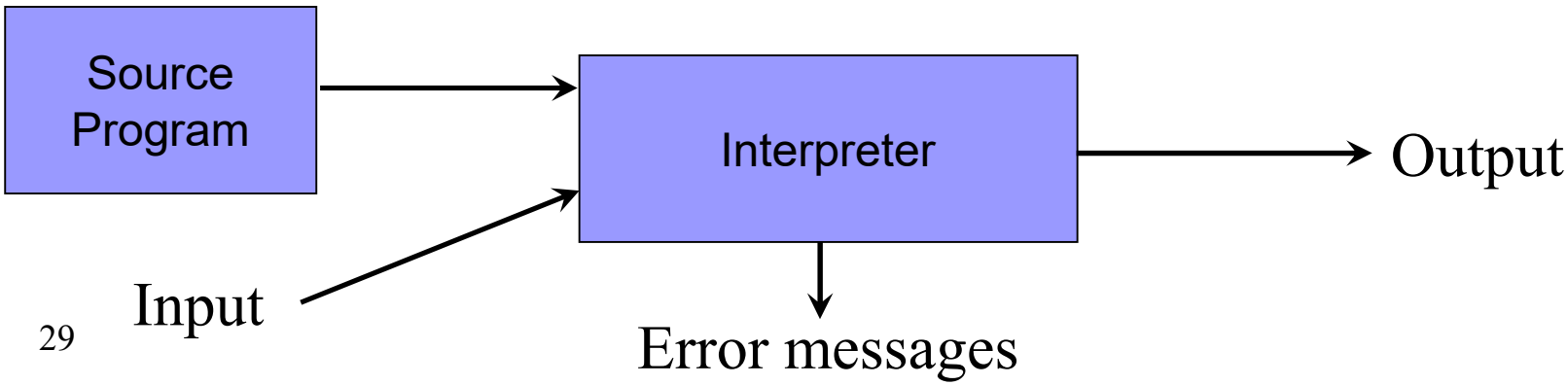


Compilers (cont'd)

- 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.

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.
- Programming language like Python, Ruby use interpreters.
- Oversimplified view:

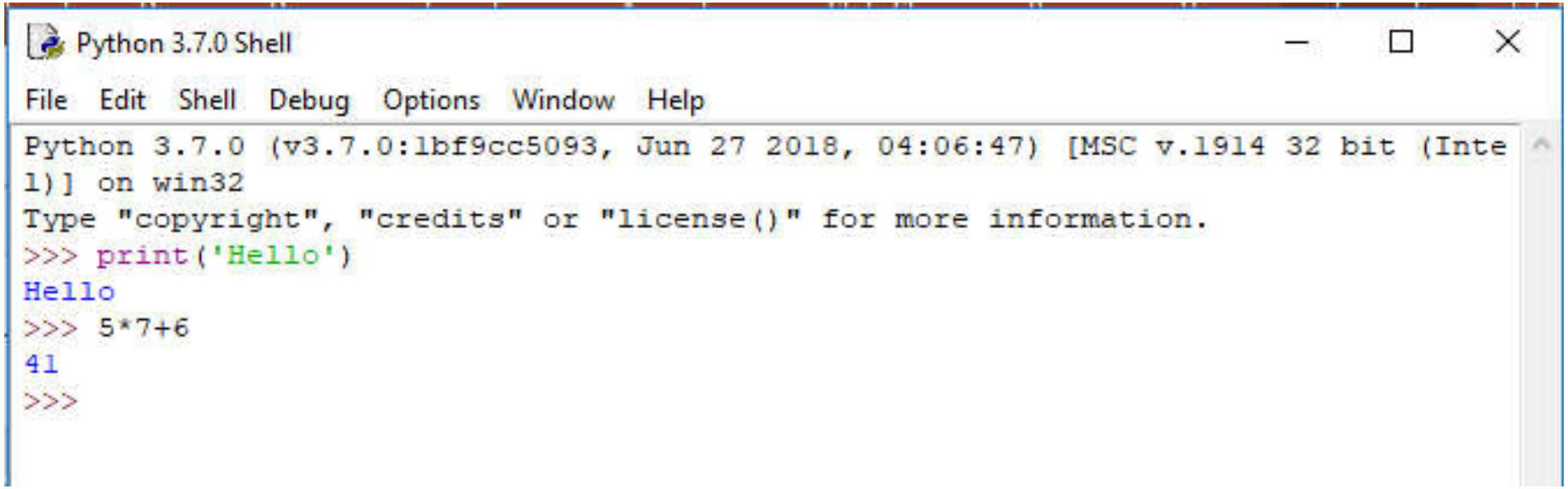


Interpreters (cont'd)

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

Python translator: interpreter

Python command window

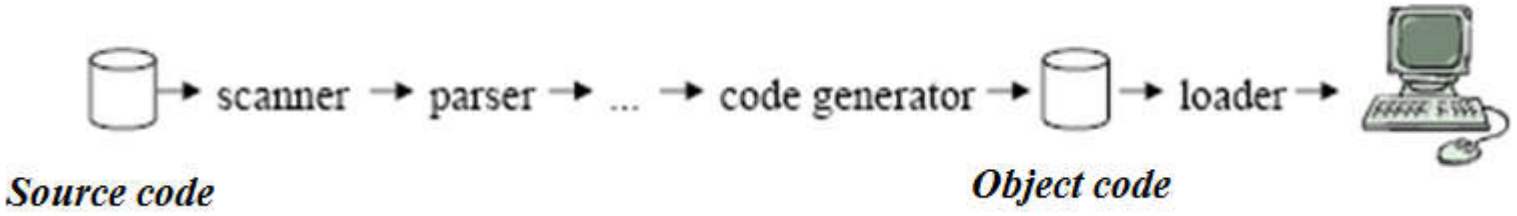


Interpreter as a part of compiler

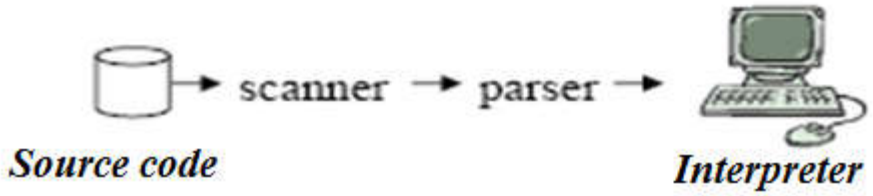
- In a compiler implementations
- The source code is compiled to a machine language for an idealized virtual machine
- The interpreter of accepts the codes and the input, produces the output.
- This technique is quite popular to make the compiler independent with the computer

Compilers and Interpreters (cont'd)

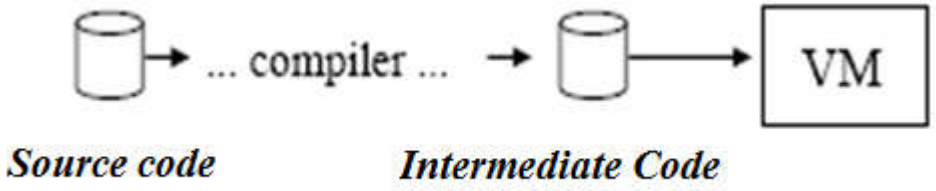
- A **Compiler** is a *program* that translates code of a *programming language* into *machine code (assembly)*



- 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