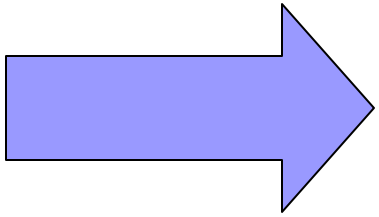
# COMPILER CONSTRUCTION

Nguyen Thi Thu Huong
Department of Computer Science-HUT

# Why study compilers?

- Experience with large-scale applications development
- Working with really big data structures
- Complex interactions between algorithms



Help you out on your next big programming project.

# More details about the course

- How do compilers work?
- How do computers work?

  (instruction set, registers, addressing modes, run-time data structures, ...)
- 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*
  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

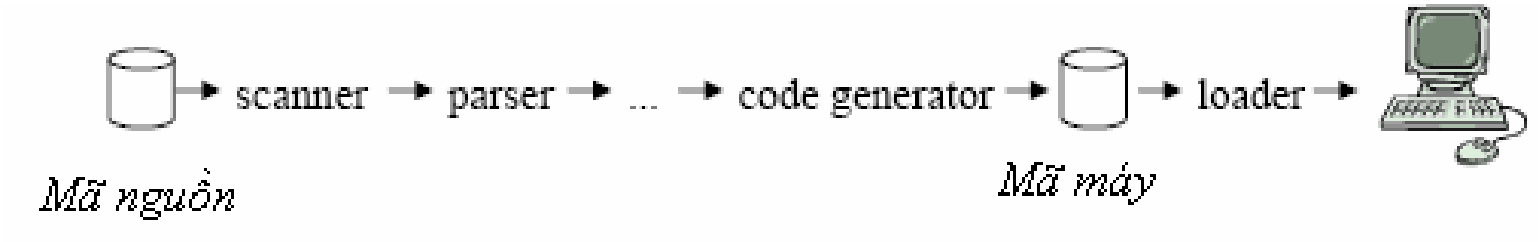
- A program that performs tasks, such as translating and interpreting, required for processing a specified programming language.For example,
  - Compiler
  - Assembler
  - Interpreter
  - Compiler - Compiler

# Compiler

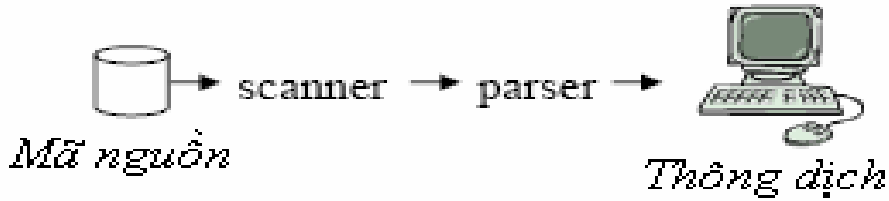
- A computer program (or set of programs) that transforms source code written in a high level language into the target language, often having a binary form known as object code.
- The most common reason for wanting to transform source code is to create an executable program.

# Compiler vs Interpreter

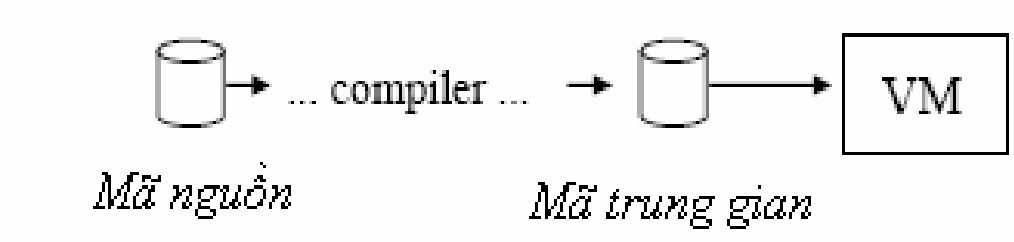
- A **Compiler** is a *program* that translates code of a *programming language* into *machine code (assembly)*




- 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