# Programming Languages

Qing Yi

Course web site:
www.cs.utsa.edu/~qingyi/cs3723

# A little about myself

- ***Qing Yi***
- *Ph.D.* Rice University, USA.
- Assistant Professor, Department of Computer Science
- **Office: SB 4.01.30**
- **Office hours: MW, 12-1pm; by appointment**
- **Phone : 458-5671**

- ***Research Interests***
- Programming Language and compiler technology
- Program analysis&optimization for high-performance computing.
- Code generation and verification of software.

# Class Objective

- Programming techniques
  - Know how to write programs in different paradigms
  - Know how to translate between different languages
- Concepts in programming languages
  - Know the concepts of typical programming languages
  - Understand how to implement programming languages (the structures of compilers and interpreters)
  - Understand trade-offs in programming language design
- Appreciate diversity of ideas
  - Critical thinking
    - **Be prepared for new problem-solving paradigms**

# General Information

- Textbook:  Concepts in Programming Languages
    - by John Mitchell, Cambridge University Press
- Reference books
    - The Little Schemer
        - by Daniel P. Friedman and Matthias Felleisen, the MIT Press.
    - Elements of ML Programming, 2nd Edition (ML97)
        - by Jerey D. Ullman, Prentice-Hall.
    - C++ Programming Language
        - by Bjarne Stroustrup, Addison Wesley.
- Prerequisites: know how to use a general purpose language
- Grading
    - Midterm and final exams: 55%
    - Homework and projects: 25% (roughly 2.5% per homework)
        - Late submissions are accepted with penalty until solution is given
    - Recitations and class participation: 15% (roughly 1% per recitation, 0.5% per class participation)
    - Problem solving: 5% (challenging projects posted periodically)
    - Extra credit projects: TBA

# Programming Paradigms

- Functional programming
  - Lisp, Scheme, ML, Haskell, …
  - Express evaluation of expressions and functions
  - Emphasize expressiveness and flexibility
    - Mostly interpreted and used for project prototyping
- Imperative programming
  - Fortran, C, Pascal, Algol,…
  - Express side-effects of statements and subroutines
  - Emphasize machine efficiency
    - Compiler optimizations (Fortran), efficient mapping to machine (C)
- Object-oriented programming
  - Simula, C++, Java, smalltalk,…
  - Emphasize abstraction and modular program organization
- Logic and concurrent programming
  - Will not be covered in this class

# Organization of class materials

- Functions and Foundations
  - Functional programming in Lisp/Scheme
  - Language syntax: compilers and interpreters
  - Language semantics: Lambda calculus
- Programming language concepts and implementation
  - Programming in ML
  - Types and type inference
  - Scopes and memory management
  - Structural control, exceptions, and continuations
- Concepts in object-oriented languages
  - C++ and Java programming
  - Modules and abstractions
  - Classes and inheritance
  - Subtyping and virtual functions

----------- final exam (comprehensive) ---------

# How to pass (or fail) this class?

1. You must work on and submit all homework assignments
2. You must attend classes/recitations and submit recitation exercises

   They prepare you to be ready for the homework assignments and exams
3. Go to my office hours (or schedule an appointment with me) if you received less than 60% from a homework

   It means you didn't understand --- figure it out before it's too late.
4. Study before exams

   Even if you think you understand everything, you may not remember them

# Languages in common use

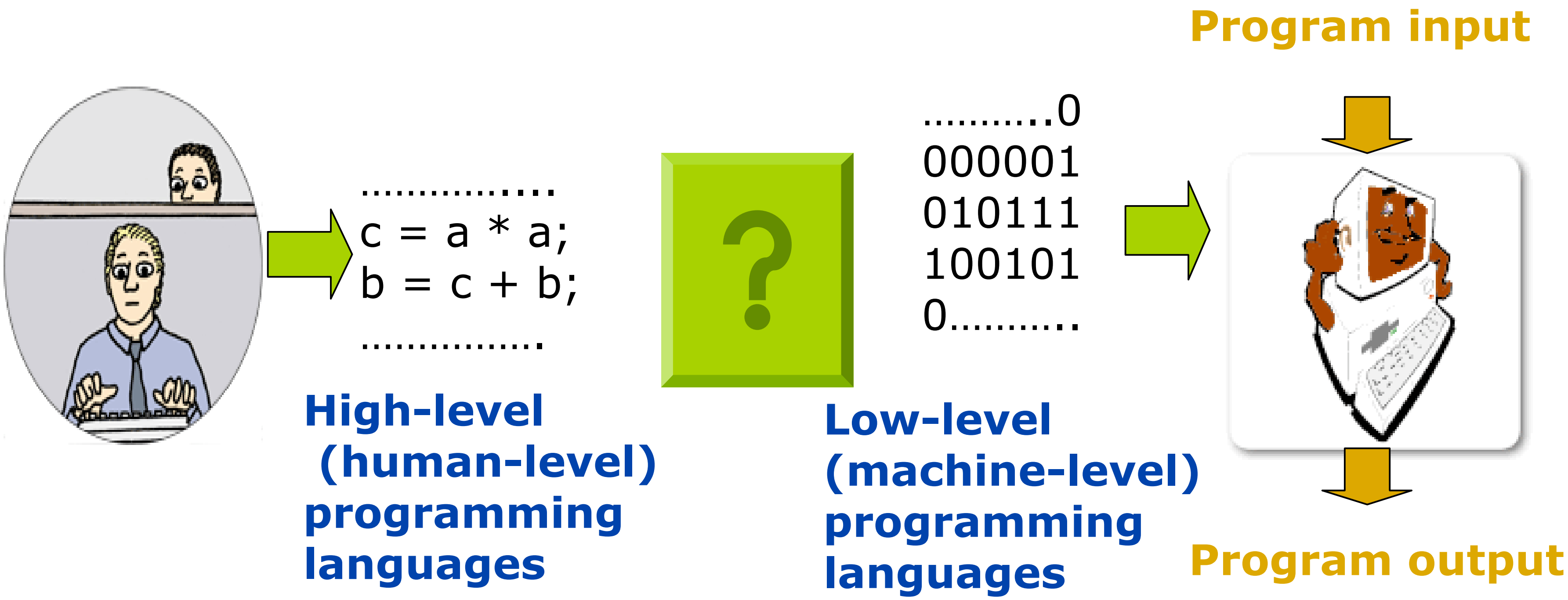
- System software and high-performance computing (e.g., weather prediction, realistic games)
  - C/C++, Fortran
- Internet and embedded systems
  - Java, C#, Ruby, Php, Javascript, xml
- System administration
  - Python, Perl, bsh, csh
- Others (non-general purpose languages)
  - Postscript (the printer language), latex (text processing), …
- What languages do you know? What paradigms do they belong?
- Check out which languages are popular
  - http://langpop.com/

# The Role of Programming Languages

- Natural languages
  - Interfaces for expressing information
    - ideas, knowledge, commands, questions, …
    - Facilitate communication between people
  - Different natural languages
    - English, Chinese, French, German, …
- Programming languages
  - Interfaces for expressing data and algorithms
    - Instructing machines what to do
    - Facilitate communication between computers and programmers
  - Different programming languages
    - FORTRAN, Pascal, C, C++, Java, Lisp, Scheme, ML, …

# Levels of Programming Languages

Two ways to implement a language: compilation vs. interpretation.



**Program input**

```
……………...
c = a * a;
b = c + b;
…………….
```

**High-level (human-level) programming languages**

?

```
………..0
000001
010111
100101
0……….
```

**Low-level (machine-level) programming languages**

**Program output**

Some languages are higher level than others, why?
(Readability, programmability, maintainability)

# Benefits of high-level languages

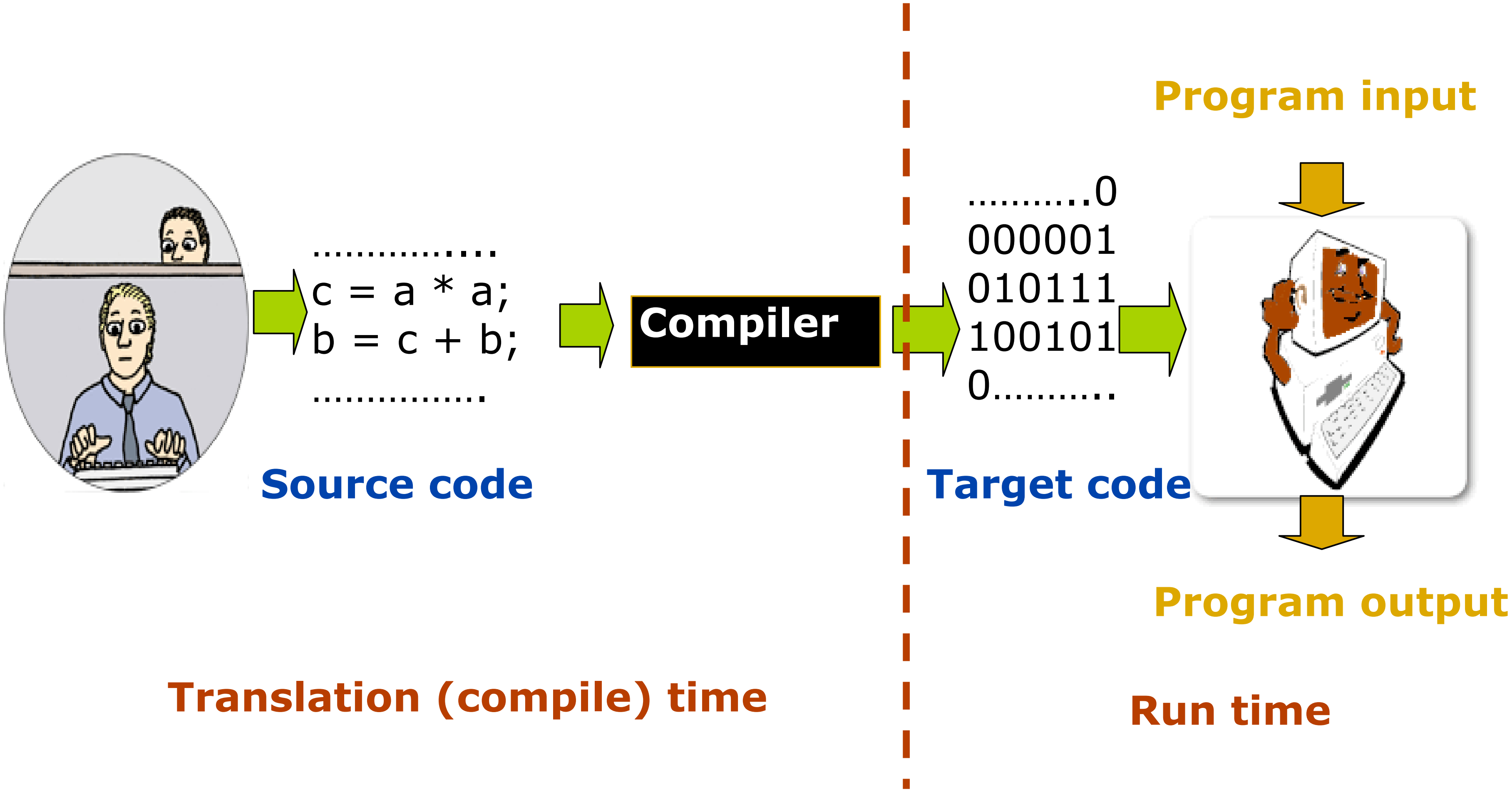
✓ Developer productivity
  - Higher level mechanisms for
    - Describing relations between data
    - Expressing algorithms and computations
  - Error checking and reporting capability

✓ Machine independence
  - Portable programs and libraries

✓ Maintainability of programs
  - Readable notations
  - High level description of algorithms
  - Modular organization of projects

X Machine efficiency
  - Extra cost of compilation / interpretation

# Implementing programming languages Compilation



Program input

```
……………...
c = a * a;
b = c + b;
…………….
```

Compiler

```
………..0
000001
010111
100101
0………..
```

Source code

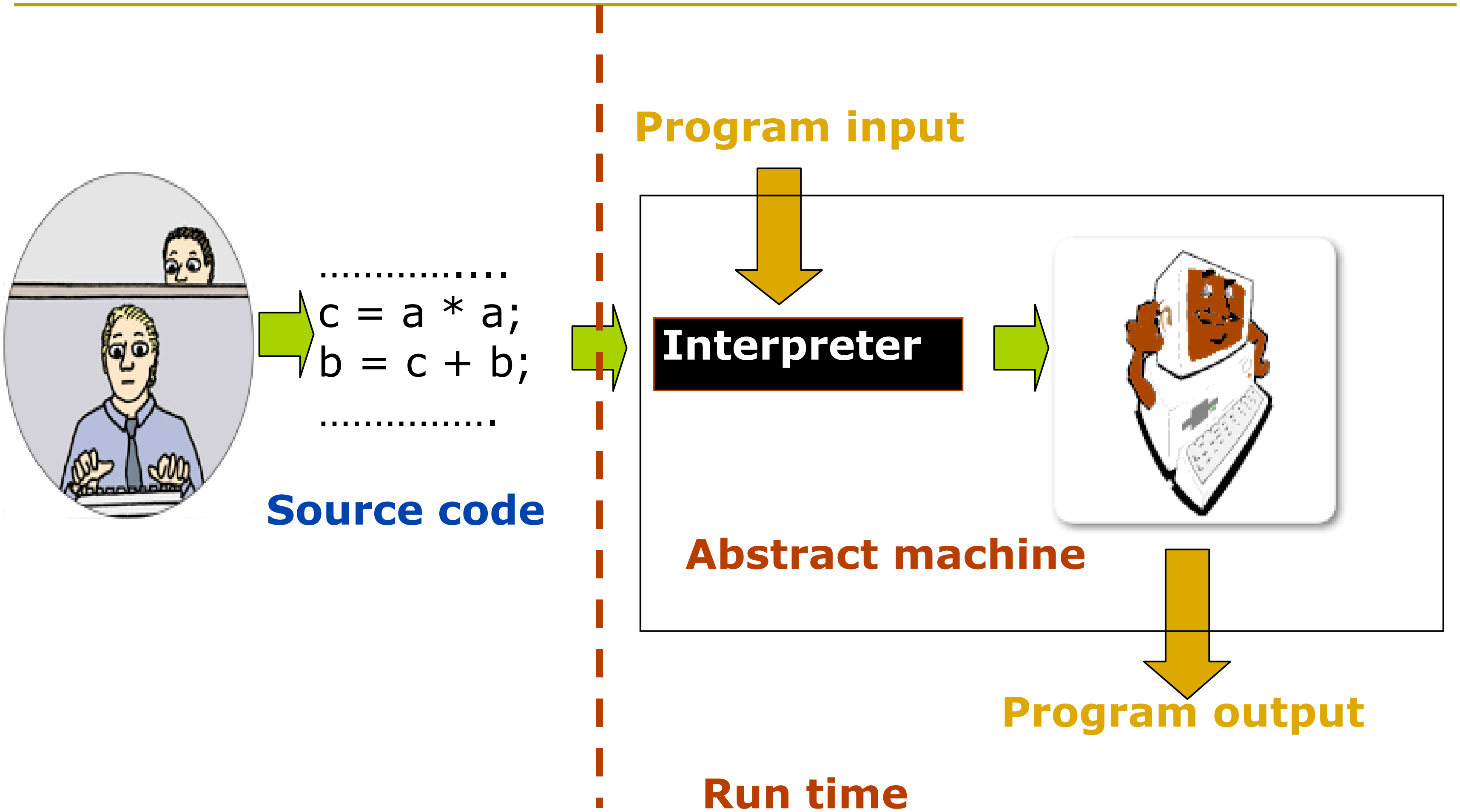
Target code

Program output

Translation (compile) time

Run time

# Implementing programming languages Interpretation



```
………………...
c = a * a;
b = c + b;
……………….
```

Source code

Program input

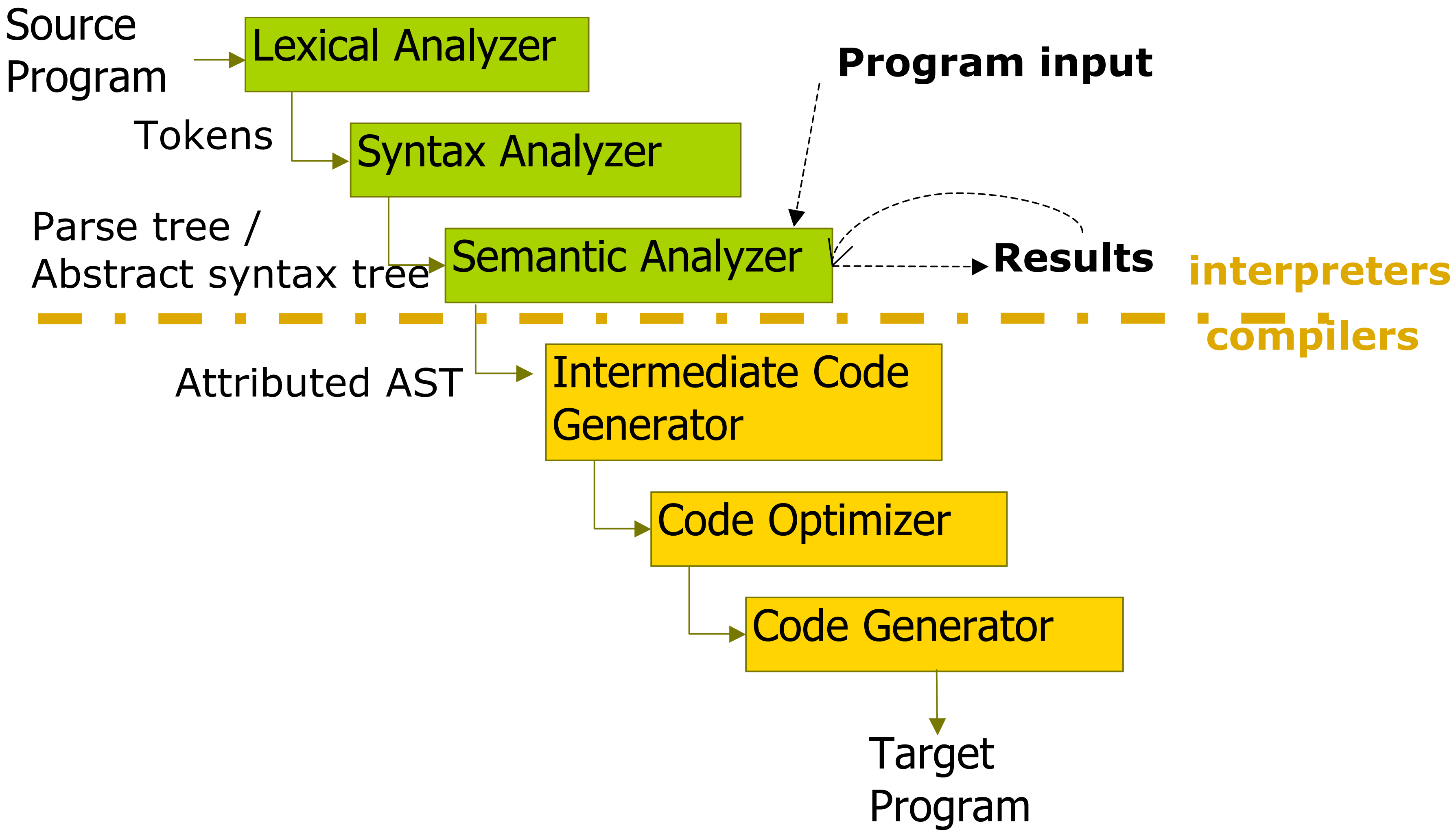
Interpreter

Abstract machine

Program output

Run time

# Compilers vs. Interpreters



# Compilers and Interpreters Efficiency vs. Flexibility

- Compilers
  - Translation time is separate from run time
    - ✓ Compiled code can run many times
    - ✓ Heavy weight optimizations are affordable
    - ✓ Can pre-examine programs for errors
    - x Static analysis has limited capability
    - x Cannot change programs on the fly
- Interpreters
  - Translation time is included in run time
    - x Re-interpret every expression at run time
    - x Cannot afford heavy-weight optimizations
    - x Discover errors only when they occur at run time
    - ✓ Have full knowledge of program behavior
    - ✓ Can dynamically change program behavior

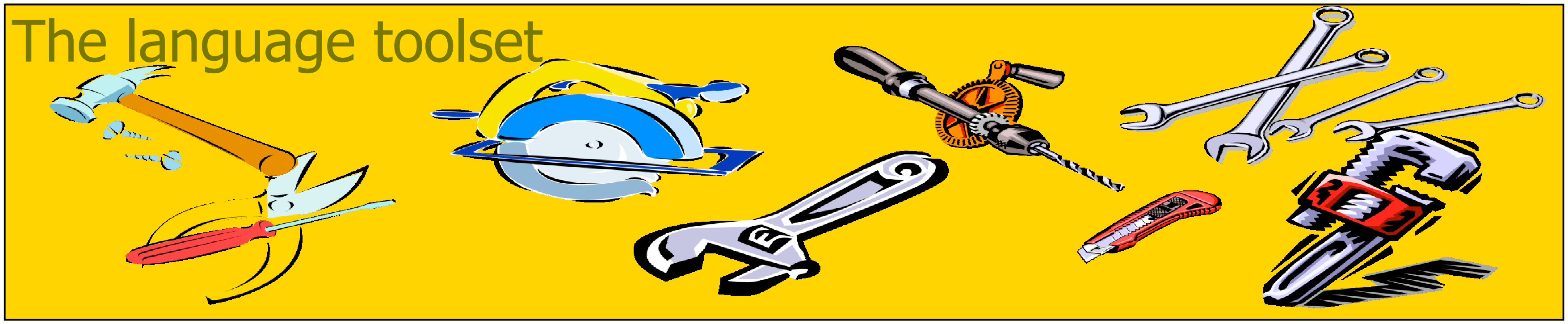
# The Power of Programming languages

- A function f is computable if for every input x
  - P(x) halts; and
  - If f(x) is defined, P(x) outputs f(x)
- Some functions are not computable
  - The halting problem
    - Given a program P that requires exactly one string input and given a string x, determine whether P halts on input x
- Terminology: **partial recursive functions**
  - Recursive functions that may be partially defined (undefined for some input values)
    - Error termination: division by zero (3/0 has no value)
    - Non-termination: f(x) = if x=0 then 1 else f(x-2)
- All programming languages are Turing complete
  - All express the class of partial recursive functions
- Programming language implementation
  - Can report error due to undefined basic operations
  - *Cannot* report error if program will not terminate

# Which problems can you solve to perfection via programming?

- Automatic translation from English to French
- A semantic query interface for the web
- Automatic translation from C++ to Java
- A grade query interface for a university student database

# The choice of Programming languages



- Most successful languages are designed for a specific type of applications
  - What does your application need?
    - Symbolic evaluation, systems programming, numerical computation, …
    - Programming efficiency vs. machine efficiency
- What languages would you choose
  - To build an embedded OS for MP3 players? A driver for your sound card? A database management system? A robot controller? A web server? ……

# Some history--- Languages that led the way

- Fortran --- the first high-level programming language
  - Led by John Backus around 1954-1956
  - Designed for numerical computations
  - Introduced variables, arrays, and subroutines
- Lisp
  - Led by John McCarthy in late 1950s
  - Designed for symbolic computation in artificial intelligence
  - Introduced higher-order functions and garbage collection
  - Descendents include Scheme, ML, Haskell, …
- Algol
  - Led by a committee of designers of Fortran and Lisp in late 1950s
  - Introduced type system and data structure
  - Descendents include Pascal, Modula, C, C++ …
- Simula
  - Led by Kristen Nygaard and Ole-Johan Dahl arround 1961-1967
  - Designed for simulation
  - Introduced data-abstraction and object-oriented design
  - Descendents include C++, Java, smalltalk …

# A New Language By You?

- Research in languages and compilers
  - Two focuses: programming productivity and machine efficiency
    - How to express high-level programming concepts (e.g., data structures and algorithms) and translate them into efficient machine implementations?
    - How to extract the most performance from machines?
  - Examples
    - How to express parallel programming effectively and efficiently?
    - How to automatically verify correctness of your programs?
    - How to automate design and implementation?
    - ……
- Thinking about graduate programs?
  - You can consider UTSA and other universities