CS571: Programming Languages

1

Course Info

- Instructor: Ping Yang

  Office: P11 (3rd floor), engineering building

  Office Hours: Wed. 10am – noon
  (Start on January. 31)

  Email: pyang@binghamton.edu

2

Course Info

- Teaching Assistants:

  Tianlin Li (1 TA)
  Office: TBA, Engineering Building
  Office Hours: (Start on Jan. 29th)
  Email: tli16@binghamton.edu

  Huiyuan Yang (0.5 TA)
  Office: TBA, Engineering Building
  Office Hours: (Start on Jan. 29th)
  Email: hyang51@binghamton.edu

3

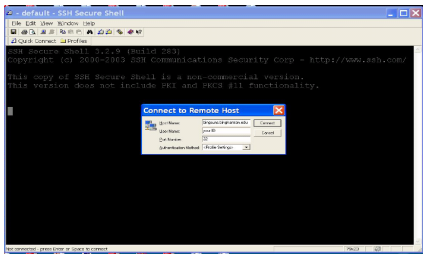
Course Info (Cont.)

- Textbook (recommended):
  - "Programming Languages: Principles and Practice" (2nd Edition), by Kenneth C. Louden
- Course website:
  http://www.cs.binghamton.edu/~pyang/cs571S18.html
- Course materials are available on the blackboard system: http://blackboard.binghamton.edu

4

Course Info (Cont.)

- Make sure that you have an account on bingsuns.binghamton.edu.
  - Windows: Download SSH secure shell client to access bingsuns
    https://cgi.math.princeton.edu/compudocwiki/index.php?title=HowTos:Connect_to_login_servers_via_ssh



5

Prerequisites

- Proficient with programming in C and (C++ or Java)
- Comfortable with writing recursive programs.
- Comfortable working and programming in the Unix environment.

6

Objectives

- To introduce you to the fundamental concepts in programming languages
- To have an in-depth understanding of different language features included in common languages such as C, C++, Java, Haskell, Prolog, Perl, PHP, and JavaScript.
- To study different language paradigms, their benefits and drawbacks.
- To understand how various language features are implemented.
- To understand the design choice and trade-offs in a language.

7

Topics

- An overview of compiler
- Basic semantics: variable, scope, pointers, parameter passing mechanisms, etc.
- Scripting Language (Perl, PHP, JavaScript)
- Functional Programming (Haskell)
- Logic Programming (Prolog)
- Object-Oriented Programming (C++, Java)

8

Grading

- Grading
  - Six Assignments: 42%
    - Assignment 1 (flex + bison + C/C++): 12%
    - Assignment 2 (basic semantics, perl): 7%
    - Assignment 3 (javascript): 10%
    - Assignment 4 (haskell): 5%
    - Assignment 5 (prolog): 5%
    - Assignment 6 (OO): 3%
- All assignments will be done by a group of 2 students.

9

Grading (Cont.)

- Grading
  - Exam 1 (tentative: March 2): 20%
    - Compiler, basic semantics, Perl
  - Exam 2 (tentative: March 30): 18%
    - Haskell, JavaScript, PHP
  - Exam 3 (May, final exam week): 20%
    - Prolog, OO

10

Grading

- Final grades will be curved over the entire class
  - A: weighted total > 92
  - A-: weighted total > 90
- If you have questions about the grading of assignments, please first contact the TA.
- If the issue has not been resolved by the TA, please email/talk to me.
- Questions regarding exams and final grades should be addressed to me.

11

Assignment/Exam Policies

- Assignments
  - Start early, ask questions early, submit on time
  - Late assignment penalty:
    - 1-6hrs: 2.5 6-12hrs: 5
- Missed exam Policy
  - There will be NO makeup exam, except in medical emergencies, when accompanied with appropriate documentation from the doctor.

12

Asking Questions

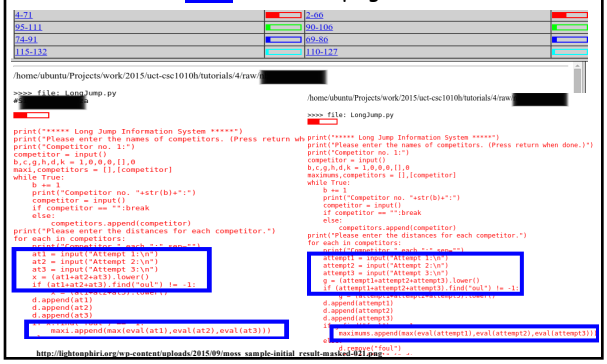
- During the class
- During office hours
- Make google your friend
- Email me/TA

Academic Integrity

- All students should follow Student Academic Honesty Code (http://www2.binghamton.edu/watson/about/honesty-policy.pdf).
- No collaborations between students in different groups
  - You may discuss the problems with students in other group, however, you must write your own codes and solutions. Discussing solutions to the problem with other groups is NOT acceptable.
  - Copying an assignment from students in another group or allowing students in other groups to copy your work may lead to an F.

Academic Integrity

- We will use Moss, to detect plagiarism.



Academic Integrity

- Use chmod 700 <directoryname> command to change the permissions of your working directories before you start working on the assignments.
- Copying materials from the Internet will be considered academic dishonesty.

Flu/Fever/Weather

- Please do not attend the class if you have flu, fever, bad cough, or any infectious diseases
- If the weather is bad (e.g. heavy snow), please check your email before you attend the lecture.

Chapter 1.2: Introduction

What is a Programming Language?

- A programming language is a notational system for describing computation in machine-readable and human-readable form.

19

What is a Programming Language?

- A programming language is a notational system for describing computation in machine-readable and human-readable form.
- Syntax: describes what programs look like
  - Informal: an if-statement consists of the word "if" followed by an expression inside parenthesis, followed by a statement, followed by an optional else part consisting of the word "else" and another statement.
  - Usually given a formal (i.e., mathematical) definition using a context-free grammar.

    <if-statement> ::= if (<expression>) <statement>
    [else <statement>]
- Semantics: describes what programs mean.

20

Semantics

- Informal

An if-statement is executed by first evaluating its expression, and if it is true, the statement following the expression is executed. If there is an else part and the expression is false, the statement following the else is executed.

21

Semantics

- Formal
  - E.g. Operational semantics of **L = E**
    - Describes the execution steps of the system
    - **s**: state – program counter + a set of variable assignments
    - If the expression **E** in state **s** reduces to value **V**, then the program **L= E** will update the state **s** with the assignment **L → V**

$$\frac{\langle E, s\rangle \Rightarrow V}{\langle L \ = E, \ s\rangle \longrightarrow (s \uplus (L \mapsto V))}$$

22

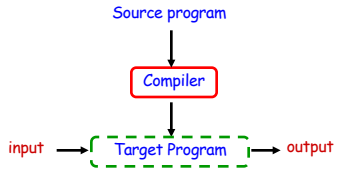
Language Translation

- Programming problems are easier to solve in high-level languages.
- High-level programming languages are not machine-readable -- we need to have a translator.

23

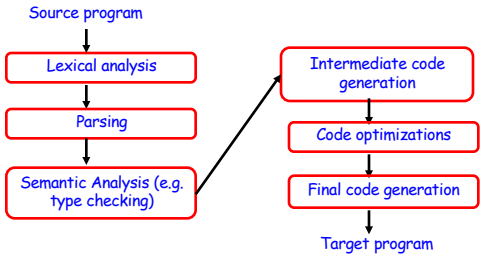
Language Translation: Compiler

- Compiler
  - translates source code into target code
  - The user may execute the target code.



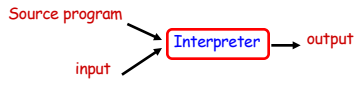
24

Language Translation: Compiler



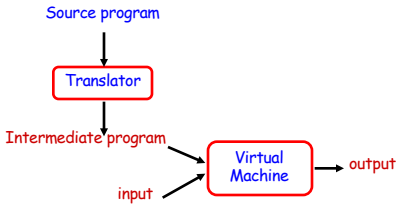
Language Translation: Interpreter

- Interpreter
  - The source code is executed directly.

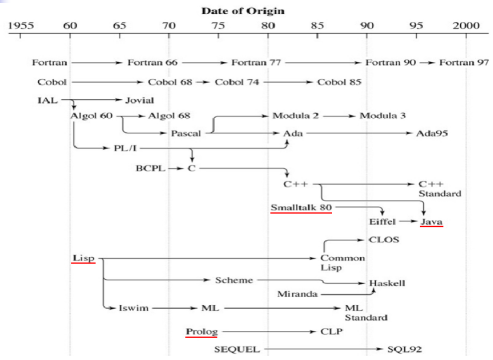


Language Translation: Interpreter

- Compiler vs. Interpreter
  - Compiler: better performance
  - Interpreter: greater flexibility and better diagnostics
- Hybrids: e.g. Java.



History of Programming Languages



Variety

- There are thousands of high-level programming languages.

Variety

- There are thousands of high-level programming languages.
  - Evolution
    - 1960-1970: goto => while loop, case statement
    - 1980: Nested block structured => object oriented
  - Special purpose
  - Personal preference

Computational paradigms

- There are thousands of high-level programming languages.
- This class:
  - Imperative
  - Functional
  - Logic
  - Object-oriented
  - Scripting

Imperative (Procedural) Languages

- Tell a computer what to do at each step.
- The hardware implementation of almost all computers is imperative
- Features:
  - The sequential execution of instructions – order of execution is critical.
  - The use of variables representing memory locations
  - The use of assignment to change the value of variables
- C, Pascal, core Ada, FORTRAN

Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
int fact(int n)
{
  int result = 1;
  while (n>0)
  {
    result = result * n;
    n = n-1
  }
  return result;
}
```

result	n
1	4

Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
int fact(int n)
{
  int result = 1;
  while (n>0)
  {
    result = result * n;
    n = n-1
  }
  return result;
}
```

Result	n
1	4
4	3

Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
int fact(int n)
{
  int result = 1;
  while (n>0)
  {
    result = result * n;
    n = n-1
  }
  return result;
}
```

result	n
1	4
4	3
12	2

Example: Imperative Languages (C)

$$factor(n) = \begin{cases} 1 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
int fact(int n)
{
  int result = 1;
  while (n>0)
  {
    result = result * n;
    n = n-1
  }
  return result;
}
```

result	n
1	4
4	3
12	2
24	1

Scripting Languages

- High-level programming languages that are interpreted at runtime
- Often used to add functionalities to Web pages
- Perl, JavaScript, PHP, Shell script

Example: Scripting Language (Perl)

$$factor(n) = \begin{cases} 1 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
sub factorial {
   if (@_[0] == 0) { 1; }
   else { @_[0] * factorial(@_[0]-1); }
}

$result = factorial(6);
print "$result\n";    # # prints "720"
```


Functional Programming Languages

- No notion of variable or assignment to variables.
- Do not worry about where things are stored – the parameters are stored in several different locations that are automatically allocated.
- Programs as collection of functions. Functions are applied to inputs that have specific values.
- Loops are replaced by recursive calls
- Application: prototyping, artificial intelligence, mathematical proof systems and so on.
- Lisp, scheme, ML, Haskell

Example: Functional Programming Languages (Haskell)

$$factor(n) = \begin{cases} 1 & if\ \ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

Example: Functional Programming Languages (Haskell)

$$factor(n) = \begin{cases} 1 & if\ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
fact x =
    if x == 0 then 1 else x * fact(x-1);
```


Example: Functional Programming Languages (Haskell)

$$factor(n) = \begin{cases} 1 & if\ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
fact x =
    if x == 0 then 1 else x * fact(x-1);

fact(4) = 4 * fact(3)
        = 4 * 3 * fact(2)
        = 4 * 3 * 2 * fact(1)
        = 4 * 3 * 2 * 1 * fact(0)
        = 4 * 3 * 2 * 1 * 1
        = 4 * 3 * 2 * 1
        = 4 * 3 * 2
        = 4 * 6
        = 24
```


Logic Programming Languages

- Programs as collections of logical statements.
- Declarative programming
  - Describe everything you know to be true about your problem and then ask questions.
- Prolog (PROgramming in LOGic).
  - Assign-once variables: any particular variable in a Prolog procedure can only ever get one value assigned to it
  - Nondeterminism: multiple definitions for the same procedure.
  - A parameter can be either input/output parameter

Example: Logic Programming Languages (Prolog)

$$factor(n) = \begin{cases} 1 & if\ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

factorial of n is 1 if n is 0
factorial of n is n times factorial of (n-1) if n is greater than 0

```
fact(N,Out):- N == 0, Out is 1.
fact(N,Out) :- N > 0,
               N1 is N - 1,
               fact(N1, Out1),
               Out is N * Out1.
```


Object-Oriented Programming Languages

- Based on a notion of an object: a collection of memory locations together with all the operations that can change the values of these memory locations.
- Encapsulation: enables the programmer to group data and the subroutines that operate on them together in one place, and to hide irrelevant details from the users.
- Java, C++, Smalltalk

Example: Object-Oriented Programming (Java)

```
public class MyInt {
  private int val;
  public MyInt(int v) {
    val = v;
  }
  public int getValue() {
    return val;
  }
  public MyInt getFact() {
    return new MyInt(fact(val));
  }
  private int fact(int n) {
    int result = 1;
    while (n > 0){
      result *= n;
      n--;
    }
    return result;
  }
}
```

$$factor(n) = \begin{cases} 0 & if\ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

Example: Object-Oriented Programming (Java)

```
public class MyInt {
  private int val;
  public MyInt(int v) {
    val = v;
  }
  public int getValue() {
    return val;
  }
  public MyInt getFact() {
    return new MyInt(fact(val));
  }
  private int fact(int n) {
    int result = 1;
    while (n > 0){
      result *= n;
      n--;
    }
    return result;
  }
}
```

$$factor(n) = \begin{cases} 0 & if\ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
/*The input is 4*/
MyInt x = new MyInt(4);
```


Example: Object-Oriented Programming (Java)

```
public class MyInt {
  private int val;
  public MyInt(int v) {
    val = v;
  }
  public int getValue() {
    return val;
  }
  public MyInt getFact() {
    return new MyInt(fact(val));
  }
  private int fact(int n) {
    int result = 1;
    while (n > 0){
      result *= n;
      n--;
    }
    return result;
  }
}
```

$$factor(n) = \begin{cases} 0 & if\ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
/*The input is 4*/
MyInt x = new MyInt(4);

/*return an object in which
the value of val is the
factorial of 4*/
x = x.getFact();
```


Example: Object-Oriented Programming (Java)

```
public class MyInt {
  private int val;
  public MyInt(int v) {
    val = v;
  }
  public int getValue() {
    return val;
  }
  public MyInt getFact() {
    return new MyInt(fact(val));
  }
  private int fact(int n) {
    int result = 1;
    while (n > 0){
      result *= n;
      n--;
    }
    return result;
  }
}
```

$$factor(n) = \begin{cases} 0 & if\ n = 0 \\ n * factor(n-1) & otherwise \end{cases}$$

```
/*The input is 4*/
MyInt x = new MyInt(4);

/*return an object in which
the value of val is the
factorial of 4*/
x = x.getFact();

/* getValue returns the value
of val*/
x1 = x.getValue();
```