Welcome to CSC324F

Programming Languages

Instructor: Marsha Chechik

Why Study Programming Languages?

- Increased capacity to express ideas (if some feature is missing - simulate it!)
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Better understanding of the significance of implementation (understand choices, understand how compiler works, judge relative efficiency of alternative constructs)
- Increased ability to design new languages
- Overall advancement of computing (choose the "right" language when possible)

CSC324F - Programming Languages

Instructor: Prof. Marsha Chechik
Office: Pratt Building Room 384
Tel.: 978-3820
E-mail: chechik@cs
Office Hours: Tu/Th 2-3 and by appt.

Class:
Lectures: Tu/Th 12-1 in MP203
Tutorials: Mondays 12-1 in
MP102, TA Iluju Kiringa, A-H
LM123, TA David Suydam, I-P
LM155, TA Ray Ortigas, R-Z
Newsgroup: ut.cdf.csc324h
Web page: www.cs/ chechik/courses97/csc324

Reading list

Required:
R. Sethi, "Programming Languages: Concepts and Constructs", 2nd edition, Addison-Wesley, 1996

Reference:
- Dybvig, "The Scheme Programming Language", Prentice-Hall, 1987
- Friedman, "Scheme and the Art of Programming", MIT Press and McGraw-Hill, 1989
- Ulman, J., "Elements of ML Programming", Prentice-Hall, 1994
- Paulson, L., "ML for the Working Programmer", 2nd edition, Cambridge University Press, 1996
- Clocksin, W. and Mellish, C., "Programming in Prolog", 4th edition, Springer-Verlag, 1994
- Bratko, "Prolog Programming for Artificial Intelligence", Addison-Wesley, 1990

Administration - Evaluation

- Assignments - 50%:
  - Assignment 1 (Scheme) - 15%
  - Assignment 2 (ML) - 15%
  - Assignment 3 (Prolog) - 20%
- Midterm Test (November 4) - 15%
- Final Exam - 35% (must get 30% to pass the course)

Several homeworks will be assigned (but not collected) towards the end of the semester.

Goals

- To learn other programming language paradigms: functional and logical
- To understand language choice tradeoffs and be able to judge new languages quickly
- To see a variety of new languages including Scheme, ML and Prolog. Some examples can be in languages like C and Pascal.
- To understand notions of programming language organization

Administration - Course outline

1. Introduction
2. Programming languages
3. Scheme and untyped PLs
4. ML and typechecking
5. Prolog
6. PL organization: grammars, parameter-passing techniques, procedure invocation

Administration - CDF

- Your programs have to run on CDF
- Electronic submission means use `submit` program
- Your account is a324+4 letters of your last name. Password is your student ID number.
- Access code is 2, 3-4, 5
- Workstations are in the Annex in EA107(?), EA201, EA203.

Everything is on the Web

Course Policies

- Assignments are due in the beginning of the class. Electronic submission deadline is 11:30.
- 10% per day lateness penalty (up to a week)
- Assignment 3b cannot be submitted late
- 10% of each mark for quality of presentation, including good English

- Regrading midterms and assignments.
  - Submit your work to person responsible for this grading
  - Explain what you want regraded and why
  - We may regrade the *entire* assignment/midterm
  - Your mark *can* go down

- Distribution of information. Newsgroup
- What is an emergency?
- First assignment - available on Sunday
- Office hours - 2-3 p.m. and by appointment. No walk-ins!

Grading responsibilities

Assignment 1 (both parts)	David Suydam
Assignment 2 (both parts)	Ray Ortigas
Assignment 3 (both parts)	Iluju Kiringa
Midterm	Marsha Chechik

- Send e-mail about assignments to TAs
- Send e-mail about course material to instructor
- TAs will hold office hours before/after the due date. Watch the newsgroup!

Introduction to Programming Languages

Chapter 1 of Sethi.

"Programming language is a means for communicating with the computer to tell it what to do."

"Different languages reflect different views on how to use a computer to solve a problem"

Outline:
- Generations of languages
- Brief history
- Comparing, designing and evaluating languages
- Language paradigms

Generations of Languages - Machine Language

"Add numbers in locations 10 and 11 and store result in location 12"

```
00000010101111001010
00000010111111001000
00000011001110101000
```

von Neumann machine, 1946

- Every operation is specified in terms of actual numeric code
- Each operand in terms of its absolute address in machine memory or a register number (or combination)
- Basic architecture includes:
  - memory locations (bytes or words, 4K to 5Gig)
  - registers - hold one word of information (1-128 bits)
  - instruction set - numerically encoded commands

Programming requires deep knowledge of underlying machine architecture.

Generations of Languages - Assembly

- Specify each operation and operand with a symbolic name
- Assembler turns assembly code into machine code
- Can have macros as shorthand for sequences of code (macro assembler)
- Machine-dependent, i.e., features depend on hardware

Programming is similar (just as hard) to programming in machine languages

Generations of Languages - High-Level Languages

- Conventional programming languages (C, C++, Lisp, Basic, COBOL)
- Usually machine-independent but implementation-dependent:

```
int b;
printf ("b is %d", b);
```

- source code -> assembly -> machine

Benefits of HLL:
- Readable, familiar notations (If, While, Procedure)
- Machine-independent
- Availability of program libraries
- Possibility of syntax checking and more thorough type checking, helping to detect errors

The programmer specifies instructions that a language processor understands and executes.

Generations of Languages - 4GL

- Very high-level languages
- Declarative languages to state *what* is to be done rather than *how* to do it.
- Report-generating languages, database processing languages, Prolog

This is the future of programming.

(Brief) History of PLs

40-50	machine and later assembly language
mid 50s	FORTRAN (automatic programming)
60s	COBOL, Lisp and Algol (first block-structured language)
late 60s	*structured programming* (SP)

SP - technique to program so that program text helps reader understand what the program does. No GOTOs!

- COBOL - business applications
- FORTRAN - scientific computing
- PL/1 - "simple good language" - considered a failure
- Most languages are "general purpose" (ex. Pascal)

Comparing languages:

which is better?

which is cheaper?

which is better designed?

Criteria for Comparison

- Intended application area (business, numerical, simulation)
- Data structures built into the language (lists, arrays, graphs)
- Operations built into the language (pattern matching, list manipulation)
- Programming paradigm implied by the language (procedural, OO, etc)
- Style of syntax (ALGOL-like, APL-like)
- Deterministic or not
- Sequential or parallel
- Compiled or interpreted

Cost of a Programming Language

• training programmers to use the language (function of simplicity and orthogonality of the language and experience of programmers)
• writing programs in the language. Depends on its closeness in purpose to the particular application
• compiling programs in the language
• executing programs written in a language. Function of language design (run-time checking - slower)
• compilers. $$ Compilation systems or hardware are bad.
• poor reliability. Software failures are $$, for safety-critical and non-safety-critical systems
• maintaining programs - depends on readability.

Language Evaluation Criteria - Readability, Writability, Reliability

Readability - the ease with which programs can be read and understood.
- Maintenance is $$$, so readability affects program quality.
- Affected by overall simplicity, orthogonality, control statements, data types and structures, syntax considerations

Overall simplicity - KISS principle:
• Language should have small number of basic components
• Language should not have feature multiplicity (multiple ways of saying the same thing):

```
count = count+1;
count +=1;
count++;
++count
```


• Better if there is no operator overloading.

```
C = A + B
```

• Too much simplicity is also bad (e.g. Assembly language)

Orthogonality - small set of features, where all combinations are legal and meaningful. No exceptions to rules → easier to learn.

```
C: records can be returned from functions but arrays cannot.
C: an array element can be any data type except void or function.
Functional languages: functions and data treated the same way
```

Control Statements - no GOTOs, allowing structured programming.

Data Types and Structures - need to define data types and data structures in a language (booleans, records).

```
FORTRAN:
 CHARACTER (LEN=30) NAME(100)
 INTEGER AGE(100), EMPLOYEENUMBER(100)
 REALSALARY(100)
```

Syntax considerations:
• *identifier forms* - short variable names are not expressive (x1, _lr)
• *special words* - endif and endloop vs begin/end vs { }, `DO` and `END` are legal names in FORTRAN77
• *Form and meaning* - design statements so that their appearance indicates their purpose. (violations in C?)

Language Evaluation Criteria - Writability

Writability - how easily a language can be used to create programs for a chosen program domain.
- Usually same features as readability, since programmers read their code
- lots of features → easy to misuse or use wrong ones; too much orthogonality → compiler cannot catch errors

Abstraction - ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored

Expressiveness - language has convenient ways of specifying computations

count++ vs count := count + 1

for vs while in Pascal

Language Evaluation Criteria - Reliability

Reliability - a program is reliable if it performs to its specifications under all conditions.
- Readability and Writability affect Reliability. Also, typechecking, exception handling and aliasing.

Typechecking - checking for type errors in a given program, statically or dynamically.

C is weakly-typed, Pascal and Turing are strongly-typed. ML has a lot of static type checking.

Exception handling - an ability of a program to intercept run-time errors, take corrective measures, and then continue.

Ada, Java have it

C++, ML have it only for user-defined exceptions

Pascal, C, FORTRAN do not have it

Aliasing - an ability to reference the same memory cell more than one different way.
- Sometimes instead of data abstraction facilities (`union` in C)

Java, designed for reliability, has no unions or direct manipulation of pointers

Summary of Evaluation Criteria

	Readability	Writability	Reliability
Syntax design	X		X
Control structures	X	X	X
Data types & structures	X	X	X
Simplicity/Orthogonality	X	X	X
Support for abstraction		X	X
Expressiveness		X	X
Type checking			X
Exception handling			X
Restricted aliasing			X

Programming Language Paradigms

Imperative Style of Programming

COBOL, C, Basic, Pascal, Algol, Turing...

-Program is intended to tell the computer to execute a set of instructions which solve a particular problem
- Based on van Naumann's view of computers

- Program changes the contents of memory in some way
- Basic operation is iteration. Basic notion - state.

Functional Programming

Started with Lisp (LISt Processor) (1958)
Currently: Lisp, Scheme, ML, Miranda, Haskell
- Given a program, what does it evaluate to?
- All programs are sequences of function applications.
- *Signature* - type of operators and function return

Examples:

`sin(90°) = 1`	Sig.: Degrees → Int (0..1)
`age(Mary) = 15`	Sig.:
`2 + 3 = 5` or	
`add(2,3) = 5`	Sig.:
`Father(Mary) = Fred`	Sig.:

`(4 + 2 * 3)` can be represented as
`add(4, times(2,3))`
what about (10 * (3 - 2)) * 6?

Logic Programming

Prolog - PROgramming in LOGic (1972).
- "What can be inferred from the facts constituting the program?"
- Program consists of facts, rules and queries.

Facts:
```
Toronto is exciting.
It is hot here.
Clyde is an elephant.
Ann has long trunk.   Ann is an animal.
```
Rules:
```
If X is an animal and X has big ears and X
has a long trunk, then X is an elephant.
```
Queries:
```
Is Toronto exciting?
Is it cold here?
Who is an elephant?
Is Marc an elephant?
```


O-O Programming

Started with Simula (language for simulation) between 1961-67.
- Key concepts: classes and subclasses; inheritance

Example:
```
Vehicle has NumOfWheels, Weight, Price
Cars and Trucks are Vehicles
Car has NumOfDoors, NumOfPassengers
Truck has Height, Tonnage
```

- "What behaviors are supported by (active) objects constituting the program?"
- Popular languages are Smalltalk, C++ and Java.