Introduction

Introduction to Programming Languages

Objective

Primary objective today:

- Let you know what is in the course
- Let you know what assignments will be given

Course

Major components of this course:

- Syntax, semantics, language translation
- Characteristics of languages and their tradeoffs
- Functional and Logic languages

Categorization of Programming Languages

Many ways to categorize languages:

- By domain: Scientific, Business, Combination
- By paradigm: Functional, Logic, VonNeumann, Scripting, Object-Oriented
- Procedural/Imperative, Declarative/Non-Imperative
- General, Domain Specific

Declarative versus Imperative

Imperative – Describe computation in terms of statements that change a program state.

Declarative languages – Non-imperative, describe the desired results of the program without explicitly listing commands or steps that need to be carried out to achieve the results.

Declarative	Imperative
Functional - Lisp/Scheme, ML, Haskell	Von Neumann – C, Ada, Fortran
Logic, constraint based – Prolog, spreadsheets	Scripting – Perl, Python, PHP
Domain –specific languages – SQL, HTML, CSS, XSLT(Extensible style sheets)	OO – Smalltalk, Eiffel, Java, C++, C#

Many Domains

Programming languages can be for a particular domain:

- Financial
- Scientific
- Embedded
- Web
- Mobile
- Gaming
- Military

No Best Programming Language – Different Needs

- Low level – need to access hardware directly
- High level –easier to program and more reliable
- Declarative – say what you want and let system determine how
- Scripting – quick and flexible
- Special purpose – databases, web sites, web services, file manipulation

Language Success

What makes a language successful?

- easy to learn (BASIC, Pascal, LOGO, Scheme)
- easy to express things, "powerful" (C#, Java, C, Algol-68, Perl)
- easy to implement (BASIC, Forth)
- possible to compile to very good (fast/small) code (Fortran, C)
- backing of a powerful sponsor (C#, COBOL, PL/1, Ada, Visual Basic)
- wide dissemination at minimal cost (Pascal, Turing, Java)

Find GCD in C

```
int gcd (int a, int b) {
    while (a!=b) {
        if (a>b) a = a-b;
        else b=b-a;
    }
    return a;
}
```

Find GCD in Scheme

```
(define gcd
   (lambda (a b)
      (cond ((= a b) a)
                ((> a b) (gcd (- a b) b))
                 (else  (gcd  a (-b a)))
       )
    )
)
```

Find GCD in Prolog

```
gcd(A, B, GCD)  :- A = B, GCD = A.

gcd(A, B, GCD)  :- A > B, C is A-B, gcd(C, B, GCD).

gcd(A, B, GCD)  :- B > A, C is B-A, gcd(A, C, GCD).
```