Course Administration

- Survey
- Course website
  http://watts.cs.sonoma.edu/cs460f23/

- BASIC
- FORTRAN
- Pascal
- COBOL
- BPL
- Audit Reporter
- RPG
- JCL
- SNOBOL
- APL
- ALGOL
- BAL
- SAS
- SPSS
- Ada
- LISP
- C
- Logo
- QBASIC
- C++
- MFC
- HTML
- Scheme
- Java
- Action Script
- C#
- XNA
- Objective C
- SVG
- Python
Why do we study Programming Languages?

- Choosing languages
- Learning languages
- Efficient program implementation
- Designing and implementing new languages
- Expressing ideas
- Overall understanding
Influences on Language Design

- Architectures
- Domains
- Paradigms
Programming Domains

- Science and Mathematics
  - FORTRAN – FORmula TRANslator

- Business
  - COBOL – Common Business Oriented Language

- Education
  - BASIC – Beginners All-purpose Symbolic Instruction Code

- Artificial Intelligence
  - LISP, Scheme

- Systems
  - Assembly languages, C

- Interactive
  - Java, VB, C#

- Web
  - HTML, XML, CSS, SVG
Programming Paradigms

- Procedural
  - FORTRAN, COBOL, BASIC, Pascal
- Functional
  - LISP, Scheme
- Logical
  - Prolog
- Object Oriented
  - Smalltalk, Java
- Scripting
  - RPG, Java Script
- Hybrid
  - C++
Language Design Factors

- Readability
- Simplicity
- Orthogonality
- Control Structures
- Data Types/Structures
- Writability
- Reliability
- Cost
Influences on Language Design

- Architectures
  - Single CPU – single processor
  - Single CPU – multiple processors
  - Multiple CPUs

- Domains
  - Calculating devices - ForTran
  - Business applications – COBOL
  - AI - Lisp
  - Education - BASIC

- Paradigms – way in which programs are written
  - Spaghetti code – lots of GOTOs!
  - Structured programming – ALGOL
  - Procedural Programming
  - Object Oriented Programming
  - Functional Programming
  - GUI / Web Programming
  - Parallel Programming
The compilation process

- Input – a human readable source program
  - Text file
  - Conforms to a specific programming language

- Output – a machine readable target program
  - A “binary” file
  - Conforms to a specific machine architecture
Language Translation

System Libraries

Source Code

System Libraries

Translator

Compiling Messages

Target Code
Phases of Compilation

- Lexical analysis
- Syntactical analysis
- Semantic analysis
- Intermediate code generation
- Optimization
- Target code generation
Lexical Analysis

```c
int 25.5 ;
```
Language Design

- Key (reserved) words (K)
- Symbols (S)
- Literals (L)
- User defined names (U)
Lexical Analysis Exercise 1

#include <iostream>
using namespace std;

int main (int argc, char * argv [])
{
    if (argc < 3)
        exit (1);
    string cat = argv[1];
cat += argv[2];
cout << cat << endl;
return 0;
}
Lexical Analysis Exercise 2

#include <iostream>
using namespace std;

int main ()
{
    int abc123, xyz;
    cout << &abc123*.0123 << endl;
    cout << -123+456 << endl;
    cout << +123.-45.67/.89 << endl;
    cout << abc123+++xyz << endl;
    return 0;
}
C++ User defined names

- Uses?
- Rules?
- Regular expression
Regular Expressions

- **Alphabet** – the symbols that actually appear in the lexeme

- **Special symbols to define the regular expression**
  - `( )`: grouping
  - `*`: 0 or more occurrences of a pattern
  - `+`: 1 or more occurrences of a pattern
  - `|`: indicates alternatives
  - `λ`: indicates nothing (lambda)
Regular Expression for User Defined Names

- Alphabet = {_, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Regular expression?
  - (_|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z)(_|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|0|1|2|3|4|5|6|7|8|9)*
Use of Underscore (_) in User Defined Names

```cpp
#include <iostream>
using namespace std;

int main ()
{
    int _;
    float __;
    string ___;
    char ____;
    bool _____;
    cout<<_<<__<<___<<____<<_____<<endl;
    return 0;
}
```