Administrative

- Instructor
  - Richard Ladner
  - ladner@cs.washington.edu

- Class info is on the web site
  - http://www.cs.washington.edu/373
  - also known as
    - http://www.cs.washington.edu/education/courses/373/02au/
Office Hours

• Richard Ladner – 311 Sieg Hall
  › W 2-3, Th 11 - 12
• Jennifer Price – 226b Sieg Hall
  › TTh 12:30 – 1:30
• David Richardson – 226b Sieg Hall
  › MW 11 - 12
CSE 373 E-mail List

- Subscribe by going to the class web page.
- E-mail list is used for posting announcements by instructor and TAs.
Computer Lab

• Math Sciences Computer Center
  › http://www.ms.washington.edu/

• Project can be done in C++ or Java.
  › I recommend Java because the text is in Java
Assignments, Projects, Exams

• Assignments 25%
  › Due on Fridays

• Projects 25%
  › Approximately 4 programming projects

• Midterm 20%
  › Friday, November 8, 2002

• Final 30%
  › Wednesday, December 18, 2002,
    8:30 – 10:20
Class Overview

- Introduction to many of the basic data structures used in computer software
  - Understand the data structures
  - Analyze the algorithms that use them
  - Know when to apply them
- Practice design and analysis of data structures.
- Practice using these data structures by writing programs.
- Data structures are the plumbing and wiring of programs.
Goal

• You will understand
  › what the tools are for storing and processing common data types
  › which tools are appropriate for which need

• So that you will be able to
  › make good design choices as a developer, project manager, or system customer
Course Topics

- Introduction to Algorithm Analysis
- Lists, Stacks, Queues
- Search Algorithms and Trees
- Hashing and Heaps
- Sorting
- Disjoint Sets
- Graph Algorithms
Reading

• Reading
  › Chapters 1 and 2, *Data Structures and Algorithm Analysis in Java*, by Weiss
Data Structures: What?

• Need to organize program data according to problem being solved
• Abstract Data Type (ADT) - A data object and a set of operations for manipulating it
  › List ADT with operations insert and delete
  › Stack ADT with operations push and pop
• Note similarity to Java classes
  › private data structure and public methods
Data Structures: Why?

- Program design depends crucially on how data is structured for use by the program
  - Implementation of some operations may become easier or harder
  - Speed of program may dramatically decrease or increase
  - Memory used may increase or decrease
  - Debugging may be become easier or harder
Terminology

• **Abstract Data Type (ADT)**
  - Mathematical description of an object with set of operations on the object. Useful building block.

• **Algorithm**
  - A high level, language independent, description of a step-by-step process

• **Data structure**
  - A specific family of algorithms for implementing an abstract data type.

• **Implementation of data structure**
  - A specific implementation in a specific language
Algorithm Analysis: Why?

- Correctness:
  › Does the algorithm do what is intended.

- Performance:
  › What is the running time of the algorithm.
  › How much storage does it consume.

- Different algorithms may correctly solve a given task
  › Which should I use?
Iterative Algorithm for Sum

• Find the sum of the first \texttt{num} integers stored in an array \texttt{v}.

\begin{verbatim}
sum(v[ ]: integer array, num: integer): integer{
    temp_sum: integer ;
    temp_sum := 0;
    for i = 0 to num - 1 do
        temp_sum := v[i] + temp_sum;
    return temp_sum;
}
\end{verbatim}

Note the use of pseudocode
Programming via Recursion

• Write a *recursive* function to find the sum of the first \texttt{num} integers stored in array \texttt{v}.

\begin{verbatim}
sum (v[ ]: integer array, num: integer): integer {
    if num = 0 then
        return 0
    else
        return v[num-1] + sum(v,num-1);
}
\end{verbatim}
Pseudocode

• In the lectures I will be presenting algorithms in pseudocode.
  › This is very common in the computer science literature
  › Pseudocode is usually easily translated to real code.
  › This is what I’m used to.

• Pseudocode should also be used for homework
Proof by Induction

• **Basis Step:** The algorithm is correct for a base case or two by inspection.

• **Inductive Hypothesis (n=k):** Assume that the algorithm works correctly for the first k cases, for any k.

• **Inductive Step (n=k+1):** Given the hypothesis above, show that the k+1 case will be calculated correctly.
Program Correctness by Induction

• **Basis Step**: $\text{sum}(v,0) = 0$.
• **Inductive Hypothesis (n=k)**: Assume $\text{sum}(v,k)$ correctly returns sum of first $k$ elements of $v$, i.e. $v[0]+v[1]+…+v[k-1]$.
• **Inductive Step (n=k+1)**: $\text{sum}(v,n)$ returns $v[k]+\text{sum}(v,k)$ which is the sum of first $k+1$ elements of $v$. 
Algorithms vs Programs

• Proving correctness of an algorithm is very important
  › a well designed algorithm is guaranteed to work correctly and its performance can be estimated

• Proving correctness of a program (an implementation) is fraught with weird bugs
  › Abstract Data Types are a way to bridge the gap between mathematical algorithms and programs