Administrivia - Introduction

CSE 373

Data Structures
Staff

• Instructor
  › Linda G. Shapiro,
    shapiro@cs.washington.edu

• TA’s
  › Gaurav Bhaya,
    gbhaya@cs.washington.edu
  › Matthew Milcic,
    mmilcic@cs.washington.edu
Linda G. Shapiro

- Professor of Computer Science and Engineering
- Professor of Electrical Engineering
- Adjunct Professor of Medical Education and Biomedical Informatics
- Research: Computer Vision & Content-Based Image Retrieval
Web Page

- All info is on the web page for CSE 373
  - also known as
Office Hours

• Linda Shapiro – 634 CSE (Allen Center)  
  › MWF 9:30-10:30 or by appointment
• Gaurav Bhaya  – to be announced
• Matthew Milcic  – to be announced
• Third TA - to be announced
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.
• It is your responsibility to subscribe. It will turn out to be very helpful for assignments hints, corrections etc.
Computer Lab

- Math Sciences Computer Center
  - [http://www.ms.washington.edu/](http://www.ms.washington.edu/)
- Project can be done in Java or C++
  - We ordered most of the texts in Java, but there should be some in C++. 
Textbook

- *Data Structures and Algorithm Analysis in Java (or in C++)*, by Weiss

- See Web page (syllabus) for errata and source code
Grading

- Assignments and programming projects 50%
- Midterm 20%
  - Approximately Feb. 13 (not definite yet)
- Final 30%
  - 2:30-4:20 p.m. Wednesday, March 17, 2004
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

- Chapters 1 and 2, *Data Structures and Algorithm Analysis in Java/C++*, by Weiss
  - Very important sections:
    - Section 1.2.5 on proofs
    - Section 1.3 on recursion
  - Most of Chapter 2 will be seen in Lecture 4
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 num integers stored in an array v.

```plaintext
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;
}
```

Note the use of pseudocode
Programming via Recursion

- Write a *recursive* function to find the sum of the first num integers stored in array v.

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

• In the lectures algorithms will be presented in pseudocode.
  › This is very common in the computer science literature
  › Pseudocode is usually easily translated to real code.
  › This is programming language independent
• Pseudocode should also be used for homework
Proof by Induction

• **Basis Step**: The algorithm is correct for the base case (e.g. n=0) 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:** \( \text{sum}(v,0) = 0. \checkmark \)

• **Inductive Hypothesis (n=k):** Assume \( \text{sum}(v,k) \) correctly returns sum of first \( k \) elements of \( v \), i.e. \( v[0] + v[1] + \ldots + v[k-1] \)

• **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 \( v. \checkmark \)
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