Introduction

CSE 373
Data Structures
Winter 2006

Staff

- Instructor
  - Hal Perkins (perkins at cs.washington.edu)
- TA's
  - Gary Yngve (gyngve at cs.washington.edu)
  - Toby Roseman (tobyr at cs.washington.edu)
- Email is particularly good for short questions, setting up appointments, topics not suitable for class discussion list. Not so good for program debugging, grading questions, …

Web Page

- All info is on the web page for CSE 373
  (or at least will be once things are a bit further along…)
  - http://www.cs.washington.edu/373
  - also known as
    - http://www.cs.washington.edu/education/courses/373/06wi
- Look there for schedules, contact information, assignments, links to discussion boards and mailing lists, etc.

Office Hours

- Hal Perkins – 548 CSE (Allen Center)
  - MW after class + appointments
- Gary Yngve – tba
- Toby Roseman – tba

CSE 373 E-mail List

- If you are registered for the course you will be automatically registered. Otherwise, subscribe by going to the class web page
- E-mail list is used for posting important announcements by instructor and TAs
  - You are responsible for anything sent here

CSE 373 Discussion Board

- The course will have a Catalyst e-post message board
- Use
  - General discussion of class contents
  - Hints and ideas about assignments (but not detailed code or solutions)
  - Other topics related to the course
Computer Lab

- Math Sciences Computer Center
  - http://www.ms.washington.edu/
- Programming language: Java 5.0
  - Java 1.4.2 will work for most things – we’ll be specific when we need Java 5 features

Programming Tools

- DrJava, Textpad, Eclipse, whatever…
  - Also may need JavaDoc, JUnit, which are easy to access from most tools
- We’re not religious about this as long as your code is standard Java
- Sun Java and most tools are freely available on the web – easy to set up at home

Textbook


Grading

Estimated Breakdown:

- Midterms 30% (15% each)
- Final 20%
  - 2:30-4:20 pm, Wednesday, March 15
- Assignments 50%
  - Weights may differ to account for relative difficulty of assignments
  - Assignments will be a mix of shorter written exercises and longer programming projects

Deadlines & Late Policy

- Assignments generally due Thursday evenings via the web
  - Exact times and dates will be given for each assignment
- Late policy: NONE
  - As in, no late assignments accepted
    (Talk to the instructor if something truly outside your control causes problems here)

Academic (Mis-)Conduct

- You are expected to do your own work
  - Exceptions (group work), if any, will be clearly announced
- Sharing solutions, doing work for or accepting work from others will be penalized
- Integrity is a fundamental principle in the academic world (and elsewhere) – we and your classmates trust you; don’t abuse that trust
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

Background

• Prerequisite is CSE 143
• Topics you should have a basic understanding of:
  › Variables, conditionals, loops, methods (functions), fundamentals of defining classes and inheritance, arrays, single linked lists, simple binary trees, recursion, some sorting and searching algorithms, basic algorithm analysis (e.g., $O(n)$ vs $O(n^2)$ and similar things)
• We can fill in gaps as needed, but if any topics are new, plan on some extra studying

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.

sum(v[], 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.

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

Pseudocode

• In the lectures algorithms will (often) be presented in "pseudocode".
  › Common in the computer science literature
  › Pseudocode is usually easily translated to real code.
  › Independent of particular programming language
  › Informal but precise: there is no "official" language definition for pseudocode

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