CSE 373 - Data Structures

April 1, 2002

http://www.cs.washington.edu/373

Administrative

- Instructor
  - › Doug Johnson
  - › djohnson@cs.washington.edu
- All class info is on the web site
  - › http://www.cs.washington.edu/373
  - › also known as
    - http://www.cs.washington.edu/education/courses/373/02sp/

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 using these data structures by writing programs using the Data Type o' the Week

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

Readings and References

- Reading
  - Chapter 1, *Data Structures and Algorithm Analysis in C*, by Weiss
- Other References
  - Sections 1-3, *Pointers and Memory*, by Parlante

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

Algorithms Analysis: What?

- What is an algorithm?
  - › A sequence of steps (a “program”) that accomplishes a task
- Many different algorithms may correctly solve a given task
  - › but will it be within this lifetime?
  - › will it require gigabytes of main memory?

Algorithm Analysis: Why?

- Understand the mathematical fundamentals needed to analyze algorithms
- Learn how to compare the efficiency of different algorithms in terms of running time and memory usage
- Study a number of standard algorithms for data manipulation and learn to use them for solving new problems

A Simple Function

- Find the sum of the first **`num`** integers stored in an array **`v`**.

```
int sum( int v[ ], int num){
    int temp_sum, i;
    temp_sum = 0;
    for ( i = 0; i < num; i++ )
        temp_sum += v[i] ;
    return temp_sum;
}
```


Programming via Recursion

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

```
int sum ( int v[ ], int num) {
    if (num == 0)
        return 0;
    else
        return v[num-1] + sum(v,num-1);
}
```

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:** sum(v,0) = 0. ✓
- **Inductive Hypothesis (n=k):** Assume 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):** sum(v,n) returns **`v[k]+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

Readings and References

- Reading
  - › Chapter 1, *Data Structures and Algorithm Analysis in C*, by Weiss
- Other References
  - › Sections 1-3, *Pointers and Memory*, by Parlante