CS200 - Class Overview

Spring 2009

Instructor: Asa Ben-Hur
Teaching Assistants: Mike Hamilton, Nick Parrish

CS200 structure

- Quizzes & Class Participation: “are you with us?”
- Tests: “what have you learned?”
- Programming assignments: “can you implement it?”
- Written assignments: “do you understand the underlying theory?”
- Check the CS200 webpage

Class meetings

- Lectures
- Recitation
  - Help you with programming and homework assignments, reinforce material from lecture.
  - you get credit for attending and participating in recit!

Difference from CS160/161

- More freedom in how to structure your program
- Larger programs, continuing assignments
- Team programming

Grading

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming assignments</td>
<td>30%</td>
</tr>
<tr>
<td>Written assignments</td>
<td>10%</td>
</tr>
<tr>
<td>Quizzes</td>
<td>10%</td>
</tr>
<tr>
<td>Recitation (attendance + completion)</td>
<td>5%</td>
</tr>
<tr>
<td>Midterms (2)</td>
<td>20%</td>
</tr>
<tr>
<td>Final</td>
<td>25%</td>
</tr>
</tbody>
</table>

You need to have a passing grade on the exams to get a passing grade in the course

Policies

“Trust men and they will be true to you; treat them greatly and they will show themselves great.” Ralph Waldo Emerson

Be professional.
Course Goals

To understand programs at different levels
- Logical view
  - Program = Algorithms + Data Structures
  - Understand their relationship and use them correctly, efficiently
- Implementation
  - Program = Objects + Methods
  - Practice design and implementation of object-oriented programs in Java
  - Read others' code and work together to build larger programs
  - Connect theory to programming concepts

Course Goals

- An understanding of a variety of common data structures
- A practical understanding of where they are applicable

Programming Assignments

- A movie recommendation system
- Netflix users rate movies they've seen
- The system recommends movies it thinks you will like (Cinematch)
- Done on the basis of ratings by users with "similar" taste.

Programming Assignments

- Netflix Prize
  - Goal: a better Cinematch
  - $1,000,000 for anyone who can predict users' rating better than Cinematch (by 10%)
  - $50,000 progress prize each year to the team whose system shows the most improvement
  - http://www.netflixprize.com/

Some Statistics

- The data set consists of more than 100 million ratings from over 480 thousand randomly-chosen, anonymous customers on nearly 18 thousand movie titles.
- Need very efficient data structures and algorithms!
Assignment 1

- A first step towards finding users with similar taste: find users who have rated movies you have rated.
- Given a database of user ratings, say how many movies a given pair of users have in common.
- Available on the course website. Due on Feb 5th. Team programming starts on next assignment.

Starting point

- Walls & Mirrors
  - Ch. 1-5: Java programming, Principles of programming, Recursion, ADTs, Linked Lists
  - Rosen
    - Ch. 1.1-1.3, 1.5-1.7, 2.1-2.3, 3.4-3.6, 4.1-4.2, 4.5, 5.1-5.3: Logic, Functions, Integers, Recursion, Counting

Clearing Cobwebs...

- Legal? let a = 5 + (int b = 4);
- Spot the bugs:
  double [] scores = {50.2, 121.0, 35.03, 14.27};
  double mine;
  for (int in = 1; in = 4; ++in) {
    mine = mine + scores[in];
  }
- What does this do?
  public static double abc(double anArray[], int x) {
    if (x == 1) { return anArray[0];}
    else { return anArray[x-1] * abc(anArray, x-1); }

Abstract Data Types (ADTs)

- Data Abstraction
  - Think of what to do with data rather than how to store it
- Procedural Abstraction
  - Think of what a method does rather than how to do it
- ADT is collection of data together with the operations defined on it
- The OO way of thinking about data structures

Typical ADT Operations

- Create new instance
- Add data
- Remove data
- Ask questions (find values, check size…)

ADT for a List

- A list: a collection of elements indexed by an integer
- What methods should be included?
ADT for a List

- Create
  - createList()

- Add
  - add(index:integer, item:ListItemType)
  - remove(index:integer)
  - removeAll()

- Remove
  - remove(index:integer)
  - removeAll()

- Ask
  - isEmpty():boolean
  - size():integer
  - get(index):ListItemType

Specifying ADT

- Describe:
  - Methods that are visible
    - Parameters
    - Return values
    - What they expect
    - What they do

Specifications

- createList() // creates an empty list
- isEmpty():boolean // returns true if list is empty
- size():integer // returns # of items in list
- add(index:integer, item:ListItemType)
  // Inserts item at position index of a list, if
  // 1<=index<=size()+1.
  // When inserted, elements with position => index are
  // renumbered to position + 1.
  // Throws an exception when index is out of range or
  // list is full.

- remove(index:integer)
  // Removes the item at position index, if 1<=index<=size().
  // After removal, elements with position => index are
  // renumbered to position-1.
  // Throws an exception when index is out of range or
  // list is empty.

- removeAll()

- get(index):ListItemType
  // Returns the item at position index if 1<=index<=size().
  // List is not changed.
  // Throws an exception if index is out of range.

Specifications (cont.)

- remove(index:integer)
  // Removes the item at position index, if 1<=index<=size().
  // After removal, elements with position => index are
  // renumbered to position-1.
  // Throws an exception when index is out of range or
  // list is empty.

- removeAll()
  // Removes all elements from list

- get(index):ListItemType
  // Returns the item at position index if 1<=index<=size().
  // Throws an exception if index is out of range.

Implementation (cont.)

- createList() {
  return numItems;
}
- isEmpty():boolean {
  return numItems == 0;
}
- size():integer {
  return numItems;
}
- add(index:integer, item:ListItemType) {
  items = new Object[MAX_LIST];
  numItems = 0;
}
- remove(index:integer) {
  items = new Object[MAX_LIST];
  numItems = 0;
}
- removeAll() {
  items = new Object[MAX_LIST];
  numItems = 0;
}
- get(index):ListItemType {
  return new Object[MAX_LIST];
}

An Array Based Implementation

Public class ListArrayBased implements ListInterface {
  private static final int MAX_LIST = 50;
  private Object items[];
  private int numItems;
  public ListArrayBased() {
    items = new Object[MAX_LIST];
    numItems = 0;
  }
  public boolean isEmpty() {
    return numItems == 0;
  }
  public int size() {
    return numItems;
  }
  public void add(int index, int item) {
    items = new Object[MAX_LIST];
    numItems = 0;
  }
  public Object get(int index) {
    return items[index];
  }
  public void remove(int index) {
    items = new Object[MAX_LIST];
    numItems = 0;
  }
  public void removeAll() {
    items = new Object[MAX_LIST];
    numItems = 0;
  }
}

Implementation (cont.)

public int size() {
  return numItems;
}
public void removeAt(int index) {
  items = new Object[MAX_LIST];
  numItems = 0;
}
public Object get(int index) {
  return items[index];
}
public boolean isEmpty() {
  return numItems == 0;
}
public void addAt(int index, int item) {
  items = new Object[MAX_LIST];
  numItems = 0;
}
public void removeAt(int index) {
  items = new Object[MAX_LIST];
  numItems = 0;
}
public void removeAll() {
  items = new Object[MAX_LIST];
  numItems = 0;
}
public Object get(int index) {
  return items[index];
}
public boolean isEmpty() {
  return numItems == 0;
}
public void addAt(int index, int item) {
  items = new Object[MAX_LIST];
  numItems = 0;
}
public void removeAt(int index) {
  items = new Object[MAX_LIST];
  numItems = 0;
}
public void removeAll() {
  items = new Object[MAX_LIST];
  numItems = 0;
}