Exam 2 Review

CS 2308
Fall 2011

Jill Seaman

Exam 2

- Thursday, October 10
- In class, closed book, closed notes, clean desk
- 20% of your final grade
- 80 minutes to complete it
- I recommend using a pencil (and eraser)
- I will bring scratch paper.
- No calculators.

Exam Format

- 100 points total
  - Plenty of writing programs/functions/classes/code
  - Some combination of:
    - Tracing code/finding errors in code (a little)
    - Short answer (a lot)
    - Fill in the blank
  - Lots of concepts in this material

Example Programming Problems

Write a function countDigits that takes a string as an argument and outputs the number of digits it contains.

Define and implement the Time class

Example Tracing Problem

Draw (and label) a diagram of memory produced by the following code:

```
struct Node {
   int data;
   Node *next;
};

Node *head = NULL;
Node *ptr = new Node;
Node *temp;
ptr->data = 42;
temp = head;
ptr->next = head;
temp = ptr;
head = new Node;
head ->data = 55;
head ->next = temp;
```


Programming on Linux

Lecture 7

- Why split code into separate files?
- How to split up the files (what goes where)
- Header files, Header files as interfaces
- How to compile multiple files
  - g++ all of them
  - separate compilation: g++ -c ...
  - makefile, how it works

Ch.10: Strings and Things

Lectures 8 and 9

- Character testing + conversion
  - isalpha, isdigit, isupper, islower, isspace
  - toupper tolower
- C-strings
  - definition, '\0' -terminated
  - why null-terminated
- C-strings: library functions
  - strlen
  - strcpy (assignment)
  - strcmp (test, comparison)

Ch.10: Strings and Things (cont.)

Lectures 8 and 9

- Predefined string class
  - how to define and initialize instances
- operations:
  - =, <<, >>, relational ops, [n] (subscript)
- member functions
  - length()
  - size()
  - append(str)
- know how to use all these to write code

Ch.13+14: Classes

Lectures 10 thru 14

- Procedural programming
  - What it is
  - What kind of problems (handling change)
- Object oriented programming:
  - What it is
  - How it solves problems of procedural problems
  - encapsulation
  - data (or information) hiding
  - interface
  - class vs instance (object)
  - accessor (getter)/ mutator (setter): why?

Ch.13+14: Classes (cont)

Lectures 10 thru 14

- Separating specifications from implementation
  - What goes where
  - What are the advantages?
  - How to compile
- Declaring a class:
  - Members: variables and functions
  - private vs public, access rules
  - syntax: class declaration
  - syntax: member function definitions
  - How to define instances
  - How to access members

Ch.13+14: Classes (cont)

Lectures 10 thru 14

- What is stale data?
- Constructors
  - How to name, return type?
  - When are they called? what do they do?
  - Default constructor
  - passing args to constructors
- Destructors
  - what it is, how to name, when is it called?
- Overloaded constructors and member functions

Ch.13+14: Classes (cont)

Lectures 10 thru 14

- Instance vs Static Members: variables+functions
- Memberwise assignment
- Copy Constructor
  - Default copy constructor
  - When to define your own
  - When is it used? (initialization only)
- Operator overloading
  - syntax of definition, use
  - how to overload assignment, relational operators

Ch.16.1: Exceptions

Lectures 15

- How to throw an exception
- Try/catch statement (syntax, semantics)
- How to use in combination
- What happens if the thrown exception is not caught?

11.9, 13.3: Pointers to Struct, Object

Lectures 15

- How to declare, assign
- How to access members through the pointer
  - (*p).member, p->member, *(p.member)
- dynamic allocation of structures, objects, arrays of structure/objects
- When an object is deleted, destructor is called
- the “this” pointer

Ch.17: Linked Lists

Lecture 16

- Dynamically allocated list data structure
- Organization: nodes, head pointer, empty list
- Declaring a linked list datatype (class declaration)
- The operations, in general
- constructor
- appendNode
- finding the last node
- traversing a linked list.

How to Study

- Start with the slides/presentations
- Read book to understand slides
- Review assignments + solutions
- Do some exercises from the book
  - Fill-in-the-Blank (vocabulary, concepts)
  - Algorithm workbench
  - Find the Error
  - Programming Challenges (first few of these)
- Be prepared to explain some things in English.