But 1st: "Quiz"/Survey Results...

- **Most challenging** project:
  - *Project 5 (object-array MP3), 17 votes* -- really needed to understand classes and multi-file compilation; proper encapsulation overwhelming at first; didn't leave adequate time to debug; incomplete driver
  - *Project 2 (procedural MP3), 8 votes* -- procedural programming resulted in lots and lots of code
  - *Project 6 (linked-list MP3), 7 votes* -- linked lists revealed missing pointer understanding; lots of segfaults
  - Other answers: P3 (3 votes), P4 (1 vote), P7 (1 vote), EC (1 vote)
  - Never mentioned: P1.0, P1.5

- **Most fun** project:
  - *Project 6 (linked-list MP3), 10 votes* -- LLs made for easier, shorter, more elegant code; challenging; brought all the course skills together
  - *Project 1.0 (image puzzles), 8 votes* -- cool to see the result of the code, finally got to see a real application of computer science in a CS project!
  - *Project 4 (password manager), 6 votes* -- real-world application, encryption is cool, educational about passwords, new programming paradigm
  - Other answers: P2 (3 votes), P5 (3 votes), P7 (3 votes), EC (2 votes)
  - Never mentioned: P1.5, P3 [no one thinks pointers are fun?]
More "Quiz"/Survey Results...

• **Most disliked** project:
  - *Project 5 (object-array MP3), 14 votes* -- multi-file issues, not solid enough on classes and object interaction, couldn't debug; incomplete driver meant students had to design test cases
  - *Project 6 (linked-list MP3), 7 votes* -- linked lists hard to debug; soooooo sick of the MP3Player by this point!  *[Project 6 was also the winner for most-fun, however!]*
  - *Project 2 (procedural MP3), 7 votes* -- the most coding
  - Several students answered, "All the MP3Player projects!"
  - Other answers: P1.0 (1 vote), P1.5 (1 vote), P3 (4 votes), P4 (2 votes), P7 (1 vote)  *[Every project hated by somebody!]*

• **Most learned-from** project:
  - *Project 6 (linked-list MP3), 13 votes* -- draws everything together: pointers, classes, etc.; classes solid by this point so got to focus on new concept
  - *Project 5 (object-array MP3), 10 votes* -- demonstration of power of classes over procedural; the project where lacking skillset from earlier in class really caught up with you  *[Overwhelming winner of most-challenging was also 2nd-most-learned-from!]*
  - Other answers: P2 (3 votes), P3 (1 vote), P4 (4 votes)
  - Never mentioned: P1.0, P1.5, P7 (not due yet)
Final Exam

• Monday, May 9, 2 PM - 4:30 PM
• In class, pencil & paper exam
• Closed book, closed notes, no cell phones or calculators, no headphones, clean desk
• 30% of your final grade
  • (Or more, if you outperform your first midterm score)
• 150 minutes to complete the exam
• Please bring a pencil and eraser!
• All writing will be done on exam paper that I'll hand out
Exam Format

• **Comprehensive (covers entire semester)**
• 100 points total
• ~20 problems, ~10 pages
• Types of questions:
  • ~40% of total points:
    • Multiple choice
    • Fill-in-the-blank concepts
    • True/false
    • Short answer
    • Algorithm illustration
  • ~60% of total points:
    • Writing classes/functions (similar to coding assignments + 1428)
    • Filling in missing statements in partially-complete code
    • Tracing and/or debugging code
Lecture Content

• All lecture slides from the semester are fair game
• Anything from 1428 is fair game

• All lecture slides available on the Schedule page of the course website
  • Animated versions are on TRACS under Resources
Assignment Content

• All 5 written homework assignments
  • Solutions on TRACS (Resources → Written Homeworks)

• All 7 (really 8) programming assignments
  • Extra credit portions of Project 6 and 7 NOT covered!
  • Extra credit assignment (TSP) NOT covered!
  • Come see me to go over my solution to any assignment

• All 6 quizzes
  • Come to office hours to get your quizzes back from me if you want (or blank ones if you weren't in class)
Textbook Content

• Use the textbook to help you understand the slides
  • There will be no questions over material or code that is in the book but NOT in the slides!

• Covered chapters:
  • Chapters 1 - 7 (1428 Review)
  • Chapter 8.1, 8.3 (Search & Sort Algorithms)
  • Chapter 9 (Pointers & Dynamic Memory)
  • Chapter 11.1 - 11.10 (ADTs/Structs)
  • Chapter 13 (Intro to Classes)
  • Chapter 14 (More About Classes)
  • Chapter 16.2 - 16.5 (Templates/STL)
  • Chapter 17 (Linked Lists)
  • Chapter 18.1, 18.3, 18.4, 18.6 (Stacks & Queues)

• Stuff that is ONLY in lecture slides (not in textbook): Linux, Algorithms & Efficiency, Program Design & Testing, C++ Multi-File Development
Linux

- Exam will assume you have been using athena.cs.txstate.edu for 3-4 months now and are comfortable navigating and developing code in that environment!
- What is Linux?
- Linux file system

**Basic shell commands**
- `pwd`
- `ls`
- `cd`
- `mkdir`
- `rm`
- `rmdir`
- `cp`
- `mv`

- Basic file editing (**vim**, **nano**, etc.)

**Compiling (g++) and running C++ programs**
- From the command line
- Using a Makefile
CS 1428 Review

- Know how to use **functions** to modularize code and promote code reuse
  - Overloaded functions, default arguments
- Know how to iterate over and operate on **arrays** (e.g., Project 1.5, Project 2, Project 5)
  - Max value, min value, linear search for value, compute average
- Know how to define and use structures
- Know how to pass parameters by reference or value
- Know how to return data from a function via return values or **reference parameters**
- Understand **partially-filled arrays** (e.g., Projects 2, 5)
- Understand arrays of structures (e.g., Projects 1.0, 2)
- Remember the **integer vs. FP divide** trick!
  - Solutions: static_cast<double>(intVar), make one of the vars a **double** even though it's an integer number
Algorithms & Efficiency

• What is an algorithm?

• How do we analyze an algorithm's runtime efficiency?

• **Big-O notation**
  • Worst-case vs. best-case vs. average-case
  • Go from f(N) to O(N)
    • Drop non-dominant terms, drop constant factors
    • Be able to order functions by growth rate
  • Given function code, state the function's runtime efficiency in Big-O
  • Given an algorithm/operation description (e.g., linear search of an array), state the operation's runtime efficiency in Big-O
    • Search/sort, array access/traversal, linked list operations
Searching & Sorting

• Searching
  • *Linear Search*
    • Write code, or explain algorithm, or perform algorithm
  • *Binary Search*
    • I will not ask you to write code for binary search, but I may ask you to fill in blanks in partial code, and you must understand the algorithm
  • Know *best case, worst case runtime efficiency*

• Sorting
  • *Bubble Sort*
  • *Selection Sort*
    • I will not ask you to write sort code, but you should be able to explain *and perform/demonstrate the algorithms*
    • Know *best case, worst case runtime efficiency* of each
    • Know how to *count the passes and comparisons* done by each
Pointers

- **Address operator** (&)
- **Pointer variables**: what are they, how to define, how to initialize
- **Dereferencing operator** (*)
  - What it means to dereference something
- **Pointers and arrays**
  - Array name is address of array's first element
  - \texttt{array[index]} = *(array + index)
- **Arithmetic and comparison operators** on pointers
- Pointers as function parameters
  - How to pass by reference using pointer parameter
- How to use **pointers with Linked Lists**
Dynamic Memory

- new operator
  - new with arrays
  - new with objects
- delete
  - delete with arrays
  - delete with objects (and interaction with destructor)

- How to (safely) return pointers from functions
Program Design & Testing

• Test plan design
  • *Identify (and isolate) boundary conditions*

• Debugging skills
  • *Trace existing code in order to identify (simple) bugs*
  • We haven't done this before... but you've presumably done this while debugging your own assignments
  • Know how to identify common errors
    • E.g., wrong array offset, integer v. FP divide, uninitialized variables
  • Approach #1: Use the output to guide what you're looking for (e.g., What's probably wrong if you see "count = -4723764629"?)
  • Approach #2: read every line of the code, say "out loud" in English what it's doing. Look for known common problems
  • Approach #3 (if above fails): re-implement the solution without looking at the given code, then compare
ADTs & Structs

• **Abstract data types**
  • What are they? What do they define?
  • Know some examples
  • *Compare to a data structure*

• **Structs**
  • Arrays of structs
  • Pointers to structs
  • *Structure pointer operator (->)*
    • *Equivalent to (ptr).field*
  • Dynamic allocation of structs
Intro to OOP/Classes

• Procedural programming
  • What is it, what's the main problem with it

• Object-oriented programming
  • What is it, how it addresses issues with procedural prog.
  • Encapsulation, data hiding, and public interfaces
  • Class vs. object-instance

• Declaring a class (e.g., Rectangle class from HW/class)
  • Members (variables and functions)
  • Private vs. public, access rules
  • Syntax for class declaration
  • Getters and setters, const functions

• How to define instances/objects of a class
• How to access class members (dot operator)
Intro to Classes (cont.)

- Inline member function definitions
- *Constructors*
  - How to name, return type
  - When are they called? What do they do?
  - *Default constructor*
  - *Constructors with arguments*
- *Destructors*
  - What are they, how to name them, *when are they called?*
- *Arrays of objects* (Project 5)
  - Declare, initialize, and access
  - *Know how to write functions to process arrays of objects*
C++ Multi-File Development

- **Separating specifications from implementation**
  - What goes in which file
  - What needs to include what, and how
  - How to compile

- Compiling multiple object files + linking executable

- **Makefiles**
  - Dependencies
  - Compiling to object files
  - Be able to write a Makefile rule
  - How to compile using Makefile from Linux cmd prompt
More About Classes

• *Instance vs. static variables*
• Dynamic allocation of objects
• Pointers to objects (and \texttt{-} \texttt{\textgreater{}} \texttt{operator} to access members)
• *The "this" pointer*
• Default assignment operator behavior for objects
• *Copy constructors*
  • Default shallow copy behavior
  • When shallow copy is a problem
  • When/how to define your own (i.e., deep copy constructor)
  • When is the copy constructor called
• Operator overloading
  • *Syntax of function definition for \texttt{==}, \texttt{<}, and \texttt{>}*
  • *Syntax of use*
Linked Lists

• What is a *List ADT*, what is a Linked List
• Linked lists vs. arrays (benefits, disadvantages)

**Organization: nodes, head pointer, empty list**

• Not in consecutive memory locations, order defined by traversal

• Declaring a linked list class (node structure, head pointer)

• **Creating an empty list (constructor)**

• `append()`, `traversal` (e.g., `display()`), deconstructor, `insert()`, and `remove()` functions are all fair game

• **Plus functions from the last slide of the LL lecture**
Stacks & Queues

- What is a **Stack ADT**? What is a **Queue ADT**?
- What do **FIFO and LIFO** mean? Which ADT uses which?
- Know the **basic operations** of each data type:
  - Stack: push, pop, isEmpty (and when you need isFull)
  - Queue: enqueue, dequeue, isEmpty (and when you need isFull)
- Understand how to use a stack for **bracket matching** (Project 7) and **postfix-notation evaluation** (HW 5)
- Be able to list common stack and queue applications
- **Know how to implement stacks and queues using Linked Lists and Arrays**
  - Which linked list operations correspond to which stack/queue ADT ops?
  - Don't need to be able to recreate the details of the circular array Queue implementation, but understand how it works
- **Illustrate contents of stack or queue after a series of operations**
How to Prepare

• Review the slides
  • Understand all the concepts and quiz yourself (or friends)
  • Read the textbook only to increase understanding of slides

• Review and re-do the 2 midterm exams
  • ~10% of exam points cover new material (Stacks & Queues); the other ~90% was covered on previous midterms

• Re-do the written homework problems (without consulting your notes)
  • Check your work against the solutions on TRACS
  • Do the problems on the Final Exam Review Exercises slides (coming soon)

• Review the coding assignments
  • Fix your code if you never got it working

• Review the quizzes (get yours back in office hours)

• Study and discuss with others!!
Remaining Spring 2016 Office Hours

- **Wednesday 4/27**
  - 3:30 PM - 4:30 PM
- **Monday 5/2**
  - 11:00 AM - Noon
  - 3:30 PM - 5:30 PM
- **Monday 5/9**
  - 11:00 AM - 1:30 PM

- Exam grades will be up in TRACS by May 16 @ 5:00 pm
- Course grades will be available in SSB on May 17 @ noon
  - *I will be traveling and out of email contact. If you want to come see your exam or talk about your grade, please email me; I will respond when I'm back in Texas.*