ECE242
Algorithms and Data Structures,
Section 2

Danny Heap
heap@cs.utoronto.ca
978-5899
SF4306

December 13, 2002

You may prefer the HTML version of this document, at
http://www.cs.utoronto.ca/˜heap/242F02/index.html

This page is for information specific to Section 2 of ECE242. For most course-related information you should probably see the main ECE242 web page.

Announcements

- Solutions for Quiz 4, PostScript, and Quiz 4, PDF
- Solutions for Quiz 3, PostScript and Quiz 3, PDF.
- Friday December 12th: Office hour cancelled. I'll be in BA5256+BA525 at a department children's party.
- Here is my last year's Quiz 3.
- Here is a sample solution for the midterm.
- The raw midterm marks have an average of **60**, not 51 as I reported in class. Professor Voss and I will discuss how (and when) to apply a linear shift to these marks (probably upwards).
- Quiz 2 Solution a, and Solution b
- Quiz 1 Solution a, and Solution b.
- Quiz 2 sample
- Midterm: Monday November 4th, 6–8 pm, SF3201. Closed book, pencil, pen, eraser allowed.
- Quiz 2: Tuesday October 29, 15 minutes, open book, in class.
- Assignment 1 — extended until September 24th, 11:59, due to heavy load on remote.ecf. To avoid the same problem, please work on p1.ecf . . . p185.ecf
- First quiz Thursday October 3, 15 minutes, open book, in class.
- TestPolynomial.c cannot check whether the degree has been (correctly) reduced by cancellation or overflow, since to do so would mean reproducing the code that should be in your Polynomial.c, so in these cases it reports that you have the wrong degree.

- New TestPolynomial.c to check `monDiv` properly.
- Errata for Assignment 1:

```
From: heap@cs.utoronto.ca (Danny Heap)
Subject: ERRATA
Newsgroups: ut.ecf.ece242
Date: 20 Sep 2002 19:36:08 GMT

In Polynomial.h two #defines are added:

#define ZERO_DEGREE -1
#define OVERFLOW -2

... and the comment for poly_t now says that the zero polynomial has
degree==ZERO_DEGREE and coefficient NULL.

TestPolynomial.c has been modified to work properly with zero polynomials.
Also notice that you are allowed to modify TestPolynomial.c in any way
you see fit.  To use the provided test data type:
        ./TestPolynomial < fourByThree.txt

The recipe for monic division (in the handout) has the words "times
the leading coefficient of P2 (either 1 or -1)" added to point 2(a),
to make clear how to construct the quotient.  The required complexity
should be O(n^3) NOT O(n) The last sentence of "What to submit" how
says "poly2 must be monic and of degree no greater than poly1" (not
poly2!).

So, please pick up new Polynomial.h, TestPolynomial.c, and the handout itself.
```

Contact

I'm located in SF4306A, 978-5899. My office hours for ECE242 are Mondays and Fridays, 4–5pm (or by appointment).

September lecture summary

September 5

- Administration, marking scheme, don't plagiarize.
- Why "Algorithms and Data Structures" is not equivalent to "The C course."
- Where C fits between micros and cosmos.
- Begin Crash course in C, Part I, pages 1–4 (see also "Data Structures and Program Design in C," Kruse, Tondo, and Leung, 629–631).
- A "Hello world" program with a comment, an #include directive, and a `main` function.
- Declarations associate a variable name with a piece of memory.
- Declarations have types, can't mix declarations and executable statements.

- Some types differ by precision. How precise? See `<limits.h>`
- You may assign values to and evaluate stored values in a variable. the `=` sign means assignment **not** equality.
- Assignments return values. You can increment variables a few ways. Prefix and postfix increments might be subtle.
- Formatted `printf` allows you to send interesting text to standard out.

September 9

- Crash course in C, Part I, pages 5–10
- conditionals
- arrays
- loops
- characters

September 10

- Crash course in C, Part I, pages 11–16.
- Functions, call by value
- scope
- global/external variables
- automatic/static variables

September 12

- Crash course in C, Part II, pages 1–8. Also, C appendix in Kruse textbook.
- Preprocessor directives, (e.g. `#define ...`), `typedef`. See boolean.c
- Pointers, pointer arithmetic, dereference (`*`) and address (`&`)
- pointers and arrays
- call by reference
- dynamic memory allocation
- (start) common errors

September 16

- Finish Crash course in C, Part II, and begin Crash course in C, Part III.
- Pointers can be manipulated with subscripts, like arrays. See dynArray.c
- Common programming errors, see error1.c, error2.c, error3.c, and error4.c.
- Create new aggregate types that mix other data types with `struct`. See StringStruct.c

September 17

- Continue Crash course in C, Part III.
- Self-referential structures for linked lists, the `struct` tag.
- the `union` type and overlapping storage.

September 19

- Continue Continue Crash course in C, Part III
- `union` type, see figure.h and figure.c.
- Separate compilation, using makefiles and header files, see makefile, TestFigure.c, and figure.h
- using the debugger, `print`, `next`, and `step` commands.

September 23

- Finish Crash course in C, Part III.
- Debug a linked list using GDB (online notes pp 11–13).
- Begin Crash course in C, Part IV, using `#[]define DEBUG` (see macros.c)

September 24

- Finish Crash course in C, Part IV
- Functions have an address that can be stored in a function pointer, and even passed to other functions. See derivatives.c
- I/O to and from a file uses `fprintf` and `fscanf`, see fileio.c

September 26

- Begin Review of recursion
- The `factorial` function is a lame excuse for recursion — it's more natural to code it iteratively.
- The `fib` function is slightly more natural to code recursively, but inefficient (unless you keep track of intermediate values).
- The Tower of Hanoi is natural to code recursively, and you can't beat it's complexity: $O(2^n)$ operations for n rings.

September 30

- Finish Review of recursion
- Why Tower of Hanoi has $O(2^n)$ complexity.
- Tail recursion.
- Begin Modularity
- Difference between public interface (*.h file) and implementation (*.c file).
- Benefits of this division of files
- How `#include headerfile.h` works
- Data abstraction example: stack.

October lecture summary

October 1

- Begin Software Engineering
- Running example: CheeseTest.c started as a `main()` function, with a long comment (in English) describing the desired solution. By step-wise refinement, needed functions were filled in. A stack ADT was first declared in intStack.h and then implemented in intStack.c

October 3

- Begin Ordered and Unordered List ADT
- An ADT is a data structured plus related operations.
- Operations for a list ADT
- Define big-O notation.
- Time efficiency of some list operations.

October 7

- Continue Ordered and Unordered List ADT.
- Queues and stacks, representations using arrays and pointers.
- Doubly linked lists, circular queues.
- Time complexity of push, pop, enter, and exit for stacks and queues.
- Improve the time complexity of enter (for a queue) by using a circular representation.

October 8

- Finish Ordered and Unordered List ADT.
- Begin Complexity, Part I
- Complexity of exchange sort
- bubble sort, and improved bubble sort
- selection sort, improved selection sort

October 10

- Continue Complexity, Part I.
- Big-Oh notation defined
- Complexity classes
- Why we care about Big-Oh
- Deriving the complexity class
- Begin Complexity, Part II.

- sequential search
- binary search
- computing A^n

October 15

- Finish Complexity, Part I.
- Begin Sorting.
- insertion sort
- Mergesort on arrays, its complexity
- Quicksort on arrays

October 17

- Continue Sorting.
- Quicksort performance.
- RadixSort, pointer and array implementation

October 21

- Finish Sorting. Radix sort performance
- Begin Trees, Part I.
- Definitions: tree, root, internal node, leaf, level, binary tree (BT), complete BT.
- Binary Search Tree (BST), inorder traversal.

October 22

- Continue Trees, Part I.
- Searching for a key in a BST
- Inserting a key in a BST
- Building a BST
- Sorting with a BST
- deleting key from a BST
- Complexity of searching a BST

October 24

- Finish Trees, Part I.
- Number of nodes in a minimum level BST of height h
- Number of nodes in a full BST of height h
- Best, worst, and average complexity of searching a full BST
- Begin Trees, Part II
- AVL trees definitions
- Insertions into AVL trees, balance factors

October 28

- Continue
- Begin Trees, Part II.
- Re-balancing an AVL tree after insertion: left rotations, right rotations, double left and double right rotations.
- AVL tree complexity analysis: N_h is the minimum number of nodes in an AVL tree of height h — use Fibonacci sequence.
- Heaps: definitions of heap order and heap shape property

October 29

- Begin Trees, Part III.
- Insertion into a heap using `bubble_up`
- Remove the largest key from a key using `bubble_down`
- Array implementation of a heap
- Complexity of insertion and deletion from a heap
- Building a heap (aka heapifying an array)
- Complexity of building a heap
- HeapSort

October 31

- Finish
- Begin Trees, Part III.
- Complexity of HeapSort
-
- Begin Trees, Part IV
- Tree traversals: PreOrder, PostOrder, InOrder
- Complexity of traversals
- Midterm review

November lecture summary

November 4

- Tree traversal algorithm: what traversal order is this?

```
Queue Q <--- empty
enter (root)
while(!empty Q)
    enter(children of next element)
    visit next element
```

- Begin Graphs, Part I
- Graph definitions
- Graph terminology
- Graph representations

November 5

- Continue Graphs, Part I
- Advantages of each graph representation
- Complexity of some operations in each representation
- Begin Graphs, Part II
- Graph traversals: BFS and DFS (generic traversal algorithm)
- Recursive DFS.

November 7

- Finish Graphs, Part II
- Begin Graphs, Part III, Minimum-Cost Spanning Trees (MCST), Prim algorithm.
- Prim algorithm for MCST optimized using heap structure.

November 11

- Continue Graphs, Part III, shortest paths algorithms.
- Dijkstra's algorithm for single source shortest paths. An example of a graph with a negative weight where Dijkstra's algorithm fails.
- Bellman-Ford algorithm for single source shortest paths. Works with negative edge weights (not for negative cycles).

November 12

- Begin Graphs, Part IV, advanced graph topics.
- DFS revisited: Depth-First Search with accounting (record arrival and departure).
- Topological sort: how to get dressed.
- Strongly Connected Components.

November 14

- Continue Graphs, Part IV.
- Difference Constraints: use shortest paths for scheduling tasks.
- Begin Hash Tables.

November 18

- Continue Hash Tables.
- Hashing with chaining.
- Some hash algorithms.

November 19

- Continue Hash Tables.
- Open addressing example. Tombstones.
- Double hashing.
- Hash performance.

November 21

- Begin Amortized analysis & Splay Trees.
- Amortized analysis of `push()`, `pop()`, `multiPop()`.
- Amortized analysis of binary counter.

November 25

- Continue Amortized analysis & Splay Trees.
- Splay trees.
- Operations zig, zag, zig-zig, zig-zag, etc.
- Splaying on a particular node.

November 26

Special topics related to Assignment 5:

- Reverse Kruskal algorithm for Minimum-Cost Spanning Tree (MCST)
- Floyd-Warshall Dynamic Programming algorithm for all-pairs shortest path.

November 28

Special topic: exhaustive search

- Look-ahead in chess
- DFS, BFS, Least-Cost Search (LCS) with mini-max
- Game of 8 with LCS.
- Eight Queen's problem

Final exam summary

The final exam lasts 2.5 hours. It will include the following types of questions:

- Short answer: (a number, word, or phrase, plus some justification).
- Longer answer: apply some idea from the course using diagrams, pseudocode, or careful sentences.
- Longest answer: extend the ideas from the course, for example implementing part of an algorithm in C, or making a complete analysis of the complexity of an algorithm.

You should prepare with these priorities:

1. Your lecture notes, assignments, midterm, online notes.
2. Course textbook
3. Old exams (lowest priority).

Use the following tactics during the exam:

- Read each question completely. Be sure you know what you're being asked to do before you begin writing an answer.
- Answer what you know well and completely.
- Don't write "filler" material for questions you don't know. Graders prefer answers that are clear but incomplete over what they perceive to be BS (Bloated Senselessness).