Eric Roberts
CS106B

Handout #26
April 24, 2009

Practice Midterm Exam #1

Review session: **Tuesday, April 28, 7:00–9:00P.M., Braun Auditorium**
Midterm #1: **Thursday, April 30, 3:15–5:15P.M., Herrin T-120**
Midterm #2: **Thursday, April 30, 7:00–9:00P.M., Braun Auditorium**

This handout is intended to give you practice solving problems that are comparable in format and difficulty to the problems that will appear on the midterm examination on Thursday, April 30. A solution set to this practice exam will be handed out on Monday along with a second practice exam.

Time and place of the exam

The midterm exam is scheduled for a two-hour block at two different times and places (note that neither of the exams is in the regular lecture room). You may take the exam at either time and need not give advance notice of which exam you plan to take. If you are unable to take the exam at either of the scheduled times, please send an e-mail message to **eroberts@cs** stating the following:

- The reason you cannot take the exam at either of the scheduled times.
- A two-hour period on Thursday or Friday at which you could take the exam. This time must be during the regular working day, and must therefore start between 8:30 and 3:00 (so that it ends by 5:00).

In order to schedule an alternate exam, I must receive an e-mail message from you by 5:00P.M. on Monday afternoon. Late requests will not be honored. Instructions for taking the midterm at an alternate time will be sent to you by e-mail on Tuesday.

Review session

There will be a general review session for the midterm on Tuesday, April 28, from 7:00–9:00P.M. in Braun Auditorium. We'll go over problems from the practice exams, but you should also bring any questions that you have.

Coverage

The midterm covers the material presented in class through Wednesday's lecture this week, which means that you are responsible for the chapters in the text through Chapter 8 ("Algorithmic Analysis").

The instructions that will be used for the actual midterm are reprinted beginning on the next page. To conserve trees, I have cut back on answer space for the practice midterm; the actual exam will have much more room for your answers and for any scratch work.

General instructions

Answer each of the questions given below. Write all of your answers directly on the examination paper, including any work that you wish to be considered for partial credit.

Each question is marked with the number of points assigned to that problem. The total number of points on the exam is 60. We intend for the number of points to be roughly comparable to the number of minutes you should spend on that problem. This leaves you with an hour to check your work or recover from false starts.

In all questions, you may include functions or definitions that have been developed in the course. First of all, we will assume that you have included any of the header files that we have covered in the text. Thus, if you want to use a `Vector`, you can simply do so without bothering to spend the time copying out the appropriate `#include` line. If you want to use functions that appear in the book that are not exported by an interface, you should give us the page number on which it appears. If you want to include code from one of your own assignments, we won't have a copy, and you'll need to copy the code you want to the exam.

Unless otherwise indicated as part of the instructions for a specific problem, comments are not required on the exam. Uncommented code that gets the job done will be sufficient for full credit on the problem. On the other hand, comments may help you to get partial credit on a problem if they help us determine what you were trying to do.

The examination is open-book, and you may make use of any texts, handouts, or course notes. You may not, however, use a computer of any kind.

Problem 1: Tracing C++ programs and big-O (10 points)

Assume that the functions `Mystery` and `Enigma` have been defined as follows:

```
int Mystery(int n) {
    if (n == 0) {
        return 1;
    } else {
        return Enigma(2, Mystery(n - 1));
    }
}

int Enigma(int n1, int n2) {
    if (n1 == 0) {
        return 0;
    } else {
        return n2 + Enigma(n1 - 1, n2);
    }
}
```

(a) What is the value of `Mystery(3)`?

(b) What is the computational complexity of the `Mystery` function expressed in terms of big-O notation, where *N* is the value of the argument `n`. In this problem, you may assume that `n` is always a nonnegative integer.

Problem 2: Vectors, grids, stacks, and queues (10 points)

Many of the figures in my books are generated by creating pictures in PostScript®, a powerful graphics language developed by the Adobe Corporation in the early 1980s. PostScript programs store their data on a stack. Many of the operators available in the PostScript language have the effect of manipulating the stack in some way. You can, for example, invoke the **pop** operator, which pops the top element off the stack, or the **exch** operator, which swaps the top two elements.

One of the most interesting (and surprisingly useful) PostScript operators is the **roll** operator, which takes two arguments: *n* and *k*. The effect of applying **roll(***n***,** *k***)** is to rotate the top *n* elements of a stack by *k* positions, where the general direction of the rotation is toward the top of the stack. More specifically, **roll(***n***,** *k***)** has the effect of removing the top *n* elements, cycling the top element to the last position *k* times, and then replacing the reordered elements back on the stack. Figure 1 at the bottom of the page shows before and after pictures for three different examples of **roll**.

Your job in this problem is to write a function

```
void Roll(Stack<char> & s, int n, int k)
```

that implements the **roll(***n***,** *k***)** operation on the character stack **s**. In doing so, you will probably find it useful to use other structures (stacks, queues, vectors, and so forth) as temporary storage. Your implementation should check to make sure that **n** and **k** are both nonnegative and that **n** is not larger than the stack size; if either of these conditions is violated, your implementation should call **Error** with the message

```
Roll: argument out of range
```

Figure 1. Three examples of the roll operator