CSCE121: Introduction to Program Design and Concepts

Practice Questions for Midterm 1

March 11, 2018

For timing purposes: the exam would have 3 of these questions, plus a few multiple choice ones as well.

Remember that when you do your midterm for real, you'll not have a compiler with you. Keep track of what information you need to look up, so that you can either memorize that information, or you can add it to the pages of notes you're permitted to bring to the exam.

Question 1: Identify the common elements of two sorted arrays

You are given two lists of numbers and are interested in the numbers common to both lists. A number x is common to $\text{list}_1 = \langle l_0, l_1, \ldots, l_p \rangle$ and $\text{list}_2 = \langle m_0, m_1, \ldots, m_q \rangle$ if and only if there exists an $0 \leq i \leq p$ and an $0 \leq j \leq q$ such that $l_i = x = m_j$. In this question you're asked to write a C++ function to identify and return the elements common to two lists provided as input. The input lists are represented as arrays of `int`s in sorted (ascending) order and there are no repeated numbers within any one array. You're given `list1` and its length `p`, `list2` and its length `q`, and are asked to return the result in `out`, which can hold at most `outcap` elements. If there are more than `outcap` common elements, return only the first `outcap` of them. Your function should return the number of items now stored in `out`.

Remember: Each list is sorted in ascending order and no number appears more than once in a list.

```
int get_list_of_common(int list1[], int p, int list2[], int q, int out[], int outcap)
```

Question 2: Letter frequencies

A classical way to break a code is to look at the frequency of symbols appearing in encoded text. In English, the three most commonly appearing letters are ‘e’, ‘t’, and ‘a’, so, for a moderately long piece of text, frequency information gives a starting point to help crack some simple codes.

Write a C++ program that reads in the text file “`input.txt`” and prints the frequency that each letter of the alphabet is used in its contents. The frequency of some letter is $\text{freq}(x) := \frac{\text{Count of } x \text{ in input}}{\text{Total number of all letters}}$. Treat uppercase and lowercase symbols as identical and ignore all other characters.

Use `char[]`s for the strings, assuming that the input has lines of no more than 100 characters in length.

Question 3: Explaining pointers

Here is some C++ code. It uses a single function, which takes a square matrix of non-negative integers as an input and processes it. This question asks you to explain, in detail, what the code does.

```
#include <iostream>
using namespace std;
const int n = 3;

int* all(int v[n][n]) { // Assumption: v contains non-negative integers
    int m = 0;
    int *ret = NULL;
    for (int i=0; i < n; i++)
        for (int j=0; j < n; j++)
            if (v[i][j] > m) {
                m = v[i][j];
                ret = &(v[i][j]);
            }
   return ret;
}

int main() {
    int t[n][n] = {{1, 23, 1}, {4, 0, 6}, {0, 12, 3}}; // Every element >= 0

    int *p = all(t);
    while (p != NULL) {
        cout << *p << endl;
        *p = 0;
        p = all(t);
    }
    return 0;
}
```

Specifically, in your explanation address the following aspects:

1. What does `all()` do? If you were to give it a better name, what might that be? Pointers are used—why are they used?
2. What does the code output? (There is a description with one or two simple sentences.) Answer this first with respect to the given input, but then generalize to explain what it does for other matrices `t`. (For guidance, work out how many lines are printed. It this always the same number, or does it depend on particular properties of the matrix?)
3. What variables, if any, undergo important state changes?

Question 4: Runs of repeated numbers of a desired length

Given a list of integers, we're interested in finding the first run of a certain length. A run is a maximal subsequence where the same number repeats. For example, in the list $\langle 1, 0, -3, 4, 6, 6, 6, 8, 8, -7, 2, 2, 2, 2\rangle$ the first run of length 2 is the pair of 8s (crucially, we don't consider the 6s because that whole run has length 3), the first run of length 3 is the triple of 6s, and the first run of length 4 is the 2s.

Write a C++ function using the declaration shown below, which takes a list of known length as input (as an array), along with a desired length to search for. It should return the position of the first run of the required length, or -1 if there are none in the input list.

```
int find_rep(int in[], int in_len, int req_len)
```

Question 5: Playing cards

In a standard deck of 52 playing cards, every card has a suit (*Clubs*, *Diamonds*, *Hearts*, or *Spades*) and is either a number card (*2–10*) or is a face card (*Jack*, *Queen*, *King*, or *Ace*).
In this question you need to:

1. Define your own type, `card_t`, to represent a playing card.
2. Write a function that determines returns true iff a `card_t` given as its argument is a face card.
3. Write a function that determines whether a hand of cards is a flush. The function should work for hands with n cards—you must pick some way to represent hands of cards. (A flush requires that the cards be of the same suit, but not form a sequence.)

Question 6: Rewrite without recursion

Here is a recursive function which computes something quite simple given a list of integers.

```
void mystery_fxn(int l[], int n)
{
   if (n < 2) return;

   mystery_fxn(l, n-1);

   int t = l[n-2];
   l[n-2] = l[n-1];
   l[n-1] = t;
}
```

The function `mystery_fxn` is used by calling it with the first argument being the array, the second being the number of elements in the array, like such:

```
int l[] = {100,200,300,400};
mystery_fxn(l, 4);
```

Examine the code and determine what `mystery_fxn` does. Then write an equivalent function without recursion. Your function should be a replacement, that is, it should handle all the same inputs as `mystery_fxn` and give an identical result.

Question 7: Reverse all words

Given a string we wish to reverse the individual words in the string. If we were given the following string: "`Here's a one-off example!  Hello world.`" then, as output, we ought to produce: "`s'ereH a ffo-eno elpmaxe!  olleH dlrow.`"
Sequences comprising of letters of the alphabet, numeric digits, the apostrophe, ampersand, and the hyphen characters all form words; any other characters do not. Complete the function `rev_words` below. You may assume that any additional functions you define can be called from it (i.e., that their declarations appear before it).

```
void rev_words(char input[])
```

Question 8: Printing all increasing sublists

If you were given the list of integers $\langle 1, 3, 2 \rangle$, then enumerating all the sublists would give the following: $\langle 1, 3, 2 \rangle$, $\langle 1, 3 \rangle$, $\langle 1, 2 \rangle$, $\langle 1 \rangle$, $\langle 3, 2 \rangle$, $\langle 3 \rangle$, $\langle 2 \rangle$, $\langle \rangle$. An increasing list $\langle x_0, x_1, \ldots, x_k \rangle$ has $x_i < x_{i+1}$ for all $i \in \{0, k-1\}$. If we wanted only the increasing non-empty sublists, then the previous eight would get trimmed to just these five: $\langle 1, 3 \rangle$, $\langle 1, 2 \rangle$, $\langle 1 \rangle$, $\langle 3 \rangle$, $\langle 2 \rangle$.

Write a C++ function to print out all the non-empty increasing sublists of a given input array.

```
void all_increasing_sublists(int list[], int n)
```

Question 9: Pass-by-reference pointers

Define a function `refs_to_pair` that has four parameters as input: it takes two `int`s, say `a` and `b`, and two pointers `min` and `max`. Write the function so that after calling it `min` points to the lesser of `a` and `b`, and `max` points to the greater of the two. Since the two pointers are updated by the function, it requires that they passed-by-reference.

Here's an example of `refs_to_pair` in action:

```
int one = 10;
int two = 20;

int *m; int *M;

refs_to_pair(one, two, m, M);
*M = (*M)*2; // double the big one
*m = (*m)/2; // halve the smaller one
cout << "Now one = " << one << ", two = " << two << endl;
```

where it ought to output "`Now one = 5, two = 40`".

END OF THE PRACTICE QUESTIONS