CSCE121: Introduction to Program Design and Concepts
Practice Questions for Midterm 3

March 27, 2019

Question 1: Printing a histogram of letters

Question 2: Checking polygon equivalence

Question 3: Explaining pointers

Question 4: Pass-by-reference pointers

Question 5: Love and love triangles

For timing purposes: the exam would have 2 or 3 of these questions, and potentially 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: Printing a histogram of letters

(This will look familiar: it was one of the questions on Midterm 2 where you had a choice. You might've done the other one, make sure you can do both!)

You are to write a function that takes in a string and produces a histogram showing the number of occurrences of each character that actually appear in the string. Here is a suitable function declaration: `void unique_chars_hist(char str[]);`
It takes in a single parameter as the input string (as an array of `char`s terminated with a '`\0`') and writes output with `cout`. On input '`mississippi!`' the function `unique_chars_hist` should print:

```
m: *
i: ****
s: ****
p: **
!: *
```

Notice that the histogram is only for characters actually in the string and that the output preserves the order that they appear in the input. (Also, though not shown, upper and lower case differ in this question.) To get full credit, your function should not make any assumptions about the length of the input string—it could be long (up to any number an `int` can store).

Question 2: Checking polygon equivalence

(This will also look familiar: it was the other option for a question to answer on Midterm 2. If you'd done the other one, make sure you can this one too!)

Here is some C++ code that defines some constants and two types:

```
#include <iostream> // Std. libary, gives us cout
using namespace std;
#define MAX 200 /* Largest polygon we consider has this many sides and points */
#define EPSILON 0.0001 /* Two floats are close enough for our purposes when
                                     their values are within this distance */
struct point_t { /* Compound data structure to represent a point */
    float x, y;
};

struct poly_t { /* A data structure to represent a polygon */
    point_t verts[MAX];
    int vert_count;
};
```

The code is intended to represent polygons, things like triangles, squares, quadrilaterals, etc. The basic idea is that to represent a polygon with k sides, you'd store k `point_t`s in the `verts` array of a `poly_t`. For examine, for a square `s` you'd have `s.verts_count` have the value 4, and `s.verts[0]`, `s.verts[1]`, `s.verts[2]`, and `s.verts[3]` be the four corners. The edges of the polygon connect neighboring vertices, so there is an edge from i to $i + 1$, and they 'wrap around' so that the first and last vertices are also connected with an edge.

1) Write a function to compare two `point_t` inputs, returning `true` if they're equal. A good declaration for it would be `bool points_equal(point_t a, point_t b);` (Make sure you use the `EPSILON` constant for comparison of floats.)

2) Now write a function `two_polys_equal`, using your `points_equal` function, to determine whether two polygons represent the same shape.

This is tricky because triangle ABC has the same shape as triangle BCA, so you code should return true for them. That means the vertex lists of two identical polygonal shapes might not start describing the shape from the same first vertex. More than that, one `poly_t` might have the vertices in clockwise order (ABC) and the other in counter-clockwise form (CAB). Your code needs to ensure that all these cases would be treated as identical.

The preceding examples with triangles suffice to make the point, but your solution should be general and work for any number of vertices. You should assume that the vertices of a given `poly_t` will be distinct. That is, the shape of a triangle ABC will not be given as a degenerate quadrilateral with repetitions in the vertex points.

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: 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.

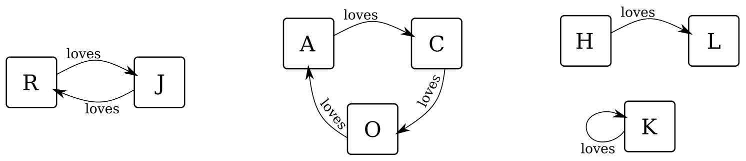
Here's an example of `refs_to_pair` in action:

```
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`".

Question 5: Love and love triangles

In this question you're asked to construct a data structure to describe romantic entanglements and love interests between people. For each person we store their name (fewer than 100 `char`s) and their age. Each person also has at most one person they are in love with. Their relationships are shown graphically below.



1. Using your type, declare and initialize two people "Romeo" (aged 16) and "Juliet" (aged 13).
2. Do the same for the love triangle "Antony" (40), "Cleopatra" (50), and "Octavia" (20).
3. And the same again for "Harold" (aged 18), "Kim" (aged 19), "Lilith" (aged 20).

END OF THE PRACTICE QUESTIONS