C M S I 3 8 6

Homework #5

Due: 2013-12-03

Submit hardcopy solutions for each of the problems. Where code is required, make sure the code is embedded within your document, but *also* provide links to the code that you have saved on ideone or codepad.

1. Frank Rubin used the following example (rewritten here in C) to argue in favor of a goto statement:

```c
int first_zero_row = -1;              /* assume no such row */
int i, j;
for (i = 0; i < n; i++) {             /* for each row */
    for (j = 0; j < n; j++) {         /* for each entry in the row */
        if (A[i][j]) goto next;       /* if non-zero go on to the next row */
    }
    first_zero_row = i;               /* went all through the row, you got it! */
    break;                            /* get out of the whole thing */
    next: ;
}                                     /* first_zero_row is now set */
```

   The intent of the code is to set `first_zero_row` to the **index** of the first all-zero row, if any, of an n × n matrix, or -1 if no such row exists. Do you find the example convincing? Is there a good structured alternative in C? In any language? Give answers in the form of a short essay. Include a good introductory section, a background section describing views on the `goto` statement throughout history, a beefy section analyzing alternatives to Rubin's problem, and a good concluding section. Talk about solutions in at least three languages.

2. For each of the following sets of strings, write functions or methods in two languages chosen from {Python, JavaScript, Java, Ruby, Clojure, C++} to return whether a given string is in the set, using regular expression matching. You will likely find, for some of the problems, that giving a finite state machine for the set is helpful. You may include a diagram of your machine in your submission but it will not affect your grade.
   a. Strings of characters beginning and ending with a double quote character, that do not contain control characters, and for which the backslash is used to escape the next character. (These are similar to, but not exactly the same as, string literals in C.)
   b. The paren-star form of comments in Pascal: strings beginning with `(*` and ending with `*)` that do not contain `*)`. Note comments "don't nest".
   c. Numbers in JSON.
   d. All non-empty sequences of letters other than "read", "red", and "real".

3. Write a **tail-recursive** function to compute the minimum value of an array in Python, C, JavaScript, and Go. Obviously these languages probably already have a min-value-in-array function in a standard library, but the purpose of this problem is for you to demonstrate your understanding of tail recursion. Your solution must be in the classic functional programming style, that is, it must be stateless. Use parameters, not nonlocal variables, to accumulate values.

4. Here's some code in some language that looks exactly like C++. It's sort of like Go, also, except the pointer types are kind of backwards. It is defining two mutually recursive types, A and B.

```
struct A {B* x; int y;};
struct B {A* x; int y;};
```

Suppose the rules for this language stated that this language used structural equivalence for types. How would you feel if you were a compiler and had to typecheck an expression in which an A was used as a B? What problem might you run into?

5. Write a program in C++, JavaScript, Python, Ruby, Scala, or Clojure that determines the order in which subroutine arguments are evaluated.

6. Consider the following (erroneous) program in C:

```
void foo() {
    int i;
    printf("%d ", i++);
}
int main() {
    int j;
    for (j = 1; j <= 10; j++) foo();
}
```

Local variable `i` in subroutine `foo` is never initialized. On many systems, however, the program will display repeatable behavior, printing `0 1 2 3 4 5 6 7 8 9`. Suggest an explanation. Also explain why the behavior on other systems might be different, or nondeterministic.

7. Consider the following pseudocode:

```
var x = 100;
function setX(n) {x = n;}
function printX() {console.log(x);}
function foo(S, P, n) {
    var x;
    if (n === 1 || n === 3) {setX(n);} else {S(n);}
    if (n === 1 || n === 2) {printX();} else {P();}
}
setX(0); foo(setX, printX, 1); printX();
setX(0); foo(setX, printX, 2); printX();
setX(0); foo(setX, printX, 3); printX();
setX(0); foo(setX, printX, 4); printX();
```

Assume that the language uses dynamic scoping. What does the program print if the language uses shallow binding? What does it print with deep binding? Why?

8. In some implementations of an old language called Fortran IV, the following code would print a 3. Can you suggest an explanation? (Hint: Fortran passes by reference.) More recent versions of the Fortran language don't have this problem. How can it be that two versions of the same language can give different results even though parameters are officially passed "the same way"? Note that knowledge of Fortran is not required for this problem.

```
      call foo(2)
      print* 2
      stop
      end
      subroutine foo(x)
          x = x + 1
          return
      end
```

9. Consider the problem of determining whether two trees have the same fringe: the same set of leaves in the same order, regardless of internal structure. An obvious way to solve this problem is to write a function `fringe` that takes a tree as argument and returns an ordered list of its leaves. Then we can say

```
def same_fringe(t1, t2):
    return fringe(t1) == fringe(t2)
```

   a. Write a straightforward version of `fringe` in Python or JavaScript.
   b. Given your straightforward code in part (a), how efficient is `same_fringe` when the trees differ in their first few leaves?
   c. If you answered part (b) correctly, you know that what you need for efficiency is laziness. Write an efficient (lazy) version of `same_fringe` in Python.
   d. Write an efficient (lazy) version of `same_fringe` in JavaScript or Go.

10. Explain what is printed under (a) call by value, (b) call by value-result, (c) call by reference, (d) call by name.

```
x = 1;
y = [2, 3, 4];
sub f(a, b) {b++; a = x + 1;}
f(y[x], x);
print x, y;
```

11. I've written a simple JavaScript queue type that does not use encapsulation. Can we achieve encapsulation using the module system in node.js? If so, implement it. If not, state why not.

12. It is certainly possible to make a Person class, then subclasses of Person for different jobs, like Manager, Employee, Student, Monitor, Advisor, Teacher, Officer and so on. But this is a bad idea, even though the IS-A test passes. Why is this a bad idea and how should this society of classes be built?

13. Write in Java, Python, JavaScript, and C++, a module with a function called `nextOdd` (or `next_odd` or `next-odd` depending on the naming conventions of the language's culture. The first time you call this subroutine you get the value 1. The next time, you get a 3, then 5, then 7, and so on. Show a snippet of code that uses this subroutine from outside the module. Is it possible to make this module hack-proof? In other words, once you compile this module, can you be sure that malicious code can't do something to disrupt the sequence of values resulting from successive calls to this function?

14. What happens to the implementation of a class if we "redefine" a field in a subclass? For example, suppose we have:

```
class Foo {
    public int a;
    public String b;
}
...
class Bar extends Foo {
    public float c;
    public int b;
}
```

    Does the representation of a *Bar* object contain one *b* field or two? If two, are both accessible, or only one? Under what circumstances? Answer for C++, Java, and Python.