Spring 2013
Programming Languages
Homework 3

- This homework includes an ML programming assignment and short answer questions. You should use Standard ML of New Jersey for the programming portion of the assignment. A link is available on NYU Classes.
- Due on Tuesday, April 16, 2013, 5:00 PM Eastern time. Submit a single ML solution file for the ML question as an attachment. Submit the short answers **inline**. The due date is also the deadline—no submissions will be accepted for credit after this date.
- Do not use imperative features such as assignment `:=`, references (keyword `ref`), or any mutable data structure, such as `Array`.
- You may use any published ML references in answering the questions. You may consult with other students to understand the questions being asked on the assignment and to understand the ML language in general. All homework solutions including algorithmic details, comments, specific approaches used, actual ML code, etc., **must** be yours alone.
- Cheating of any kind, including copying or using someone else's code, in part or in total, is strictly prohibited.
- Please see `http://www.smlnj.org/doc/errors.html` for information on common ML errors.
- There are 100 possible points. For the ML question, you will be graded based on program correctness, compliance with all requirements, use of comments, and programming style.

Hamiltonian Cycle [60] Write an ML program to solve the *Hamiltonian Cycle* problem.

Hamiltonian Cycle is a famous NP-complete problem. The problem is stated as follows: given a directed[1] graph, does there exist a cycle through the graph such that every vertex is visited exactly once?

This is called a *decision problem* because it yields a yes or no answer. If yes, the solution should also include a *witness* to the existence of a Hamiltonian Cycle (i.e., an exemplary cycle proving that it exists).

There is currently no known efficient (polynomial time) algorithm for solving the problem. Finding such an efficient solution would settle the P=NP question and undoubtedly make you a hero. However, solving the problem *inefficiently* is easy: Starting at any arbitrary vertex in the graph, perform a depth-first edge traversal, inserting into an initially empty set every vertex visited along the way. Assuming an n-vertex graph, a depth-first n-edge traversal will yield an n-member set if every vertex is visited exactly once. (To confirm a cycle, one must ensure that an edge back to the starting vertex exists too.)

There exist other methods of deciding the solution. Although they are not mentioned here, you may implement your solution differently than described here.

The method described above is also known as the *brute force* method, since it amounts to exploring the space by following every edge until the presence of a Hamiltonian Cycle is either confirmed or denied. Your assignment is to implement the Hamiltonian Cycle problem using the brute force method in ML. Depth-first traversals are naturally implemented in ML as recursive functions.

Please specifically do the following:

1. Write a `signature` and corresponding `structure` containing a set type and set operations for inserting, removing, testing membership, and returning set size. The signature should conceal implementation details. Your set must at least work on type `int`, although more graceful solutions should support sets of any type.

2. Write a `signature` and corresponding `structure` for a graph library, containing functions *at least* for adding vertices and edges to the graph, and returning the successors or predecessor of a given vertex. The library should contain a separate ML `datatype` for an edge and vertex. Each vertex can have a unique identity of type `int`. More graceful solutions should support identities of any type (a functor will be required).

3. Write an ML function `hamcycle : edge list -> edge list option` which will decide the Hamiltonian Cycle solution and produce a witness trace if it exists. More specifically, the input to `hamcycle` should be a

[1] We assume directed graphs for this assignment. Undirected graphs may be used in variations of the problem seen elsewhere.

list of edges (e.g. `[ Edge(1,2), Edge(2,4), Edge(2,5), Edge(5,1) ]`) representing the graph.

Your algorithm should infer the set of vertices from the edge list[2]. The algorithm should return the value `NONE` if no Hamiltonian Cycle exists or `SOME` list of edges as a witness to the existence of a Hamiltonian Cycle.

You can, and should, create "helper functions" as needed to implement any of the above functions. Do not use any built-in ML library functions for this assignment except for functions covered in the lecture `hd`, `tl`, `foldr`, etc. Please comment your code thoroughly.

This assignment should not require ML knowledge beyond the basic language features covered in class, although referencing the optional ML book discussed on the course page may help solidify your understanding and boost your confidence.

Advice: try to "think functionally." Do not view the problem in terms of changing the state of variables as you would in imperative programs, but rather in terms of creating new expressions from other expressions. Imperative programmers who are new to functional programming tend to over-complicate the solution with unnecessary functions and logic.

If your solution becomes unwieldy or confusing, there's probably a better way to do it. Well-written ML functions are short and concise, generally use the pattern matching style of function definitions, and should look more or less like the functions discussed on the lecture slides.

[2]Due to the simplified representation, we assume that there will not exist any vertices in the graph with no incoming or outgoing edges.

Java Modules [15] Please refer to section 7 of *Java Language Specification 7* (JLS7) in answering the following questions. The document is in the Resources folder in NYU Classes. For each, cite evidence from the standard. Although you may quote when appropriate, you must provide your own explanation. Be sure to give credit whenever quoting from a source.

1. According to JLS7, what happens if the programmer doesn't write a package declaration at the top of a file?
2. Can a standard-compliant Java implementation impose a limit of only 1 unnamed package?
3. What's the relationship between packages and compilation units?
4. Is it necessary to import a Java package before using it? That is, are Java modules open?
5. Other languages have a `using` clause—distinct from the concept of importing a module—that makes names in the module visible without qualification. Does Java have something similar?
6. What does `import static` mean? Why would one need to do this?
7. Discuss something else you learned from JLS7 regarding packages (other than something you may have learned above) that you didn't know previously.

Object Oriented Programming in C# [15] Please refer to sections 8 and 17 of the C# standard, *ECMA-334*, and answer the following questions about the C# programming language. The same rules as stated in the previous question apply.

1. What does the keyword `sealed` mean?
2. What is a `partial` declaration?
3. Can classes be nested?
4. Most C# and Java programmers would recognize the keyword `new` as relating to memory allocation. However, it's also a *class modifier*. What is a "class modifier"? What other meaning does `new` have as it relates to classes?
5. In most object-oriented languages, classes can have *fields* (also known as *data members*). Additionally, C# has something called *properties*, each of which has a "get accessor" and "set accessor." What are "properties" and how do they differ from fields?
6. Object-oriented programmers are familiar with accessibility keywords, `public`, `protected`, and `private`. However, C# has two others: `internal` and `protected internal`. What do these last two keywords mean?
7. What is a `volatile` field and why might one use it?
8. What is the difference between keywords `virtual` and `override`?

More Object Oriented Programming [10] Consider the following code below, written in a prototype OOL:

```
var obj1;
obj1.foo = 10;
var obj2 = clone(obj1);
var obj3 = clone(obj2);
obj2.bar = 20;
obj3.foo = 15;
obj1.foo = 30;
```

Assume that the program fragment above has executed. Answer the following:

1. What fields are contained locally to `obj1`?
2. What fields are contained locally to `obj2`?
3. What fields are contained locally to `obj3`?
4. To what value would `obj1.foo` evaluate, if any?
5. To what value would `obj2.foo` evaluate, if any?
6. To what value would `obj3.foo` evaluate, if any?
7. To what value would `obj1.bar` evaluate, if any?
8. To what value would `obj2.bar` evaluate, if any?
9. To what value would `obj3.bar` evaluate, if any?