Computer Science 245
Final Examination
May 8, 2007

1. a. Suppose the following class to represent a stack of characters:

```
public class Stack {
    public class Node {
        char datum;
        Node next;
        Node(char datum, Node next) { this.datum = datum; this.next = next; }
    }
    private Node top;
    public Stack() { top = null; }
    public boolean isEmpty() { return top == null; }
    public char pop() {
        char toReturn = top.datum;
        top = top.next;
        return toReturn;
    }
    public void push(char item) {
        top = new Node(item, top);
    }
}
```

Given also the following method for reversing a String:

```
    public static String reverse(String str) {

        Stack st = new Stack();

        for (int i = 0; i < str.length(); i++)

            st.push(str.charAt(i));

        String reverse = "";

        while(! st.isEmpty())

            reverse += st.pop();

        return reverse;
    }
```

Analyze the running time for this method, finding a formula and a big-oh category. You may assume instantiation, `length()`, and `charAt()` take constant time; you must figure out how to accound for the stack operations. (8 points)

b. Now suppose the following class to represent a stack of characters:

```
public class Stack {
    public class Node {
        char datum;
        Node next;
        Node(char datum, Node next) { this.datum = datum; this.next = next; }
    }
    private Node top;
    public Stack() { top = null; }
    public boolean isEmpty() { return top == null; }
    public char pop() {
        Node previous = null;
        Node current = top;
        while (current.next != null) { previous = current; current = current.next; }
        if (previous == null) top = null;
        else previous.next = null;
        return current.datum;
    }
    public void push(char item) {
        if (top == null) top = new Node(item, null);
        else {
            Node current = top;
            while (current.next != null) current = current.next;
            current.next = new Node(item, null);
        }
    }
}
```

Given also the same method for reversing a String:

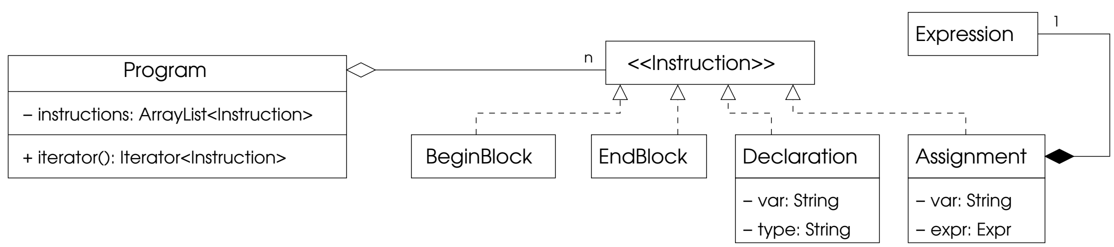
```
public static String reverse(String str) {

    Stack st = new Stack();

    for (int i = 0; i < str.length(); i++)

        st.push(str.charAt(i));

    String reverse = "";

    while(! st.isEmpty())

        reverse += st.pop();

    return reverse;
}
```

Analyze the running time for this method, finding a formula and a big-oh category.
(8 points)

2. The following question requires you to think about the design for writing a compiler or language interpreter.

At one stage in the compilation process, the compiler will scan the program to gather information on all the variables used in the program—primarily their name, their scope, and their type. One use of this information is so that the compiler can check that the variables are used correctly; that is, no undeclared variables are used, variables are used only in their scope, variables are used only in ways consistent with their declared type. The structure used to store this information is called a *symbol table*, because it is a table in which one can lookup information on symbols, that is, variables and other identifiers.

Suppose in our compiler we represent an input program with the system of classes illustrated below. A program is a sequence of (pseudo-)instructions: declarations, assignments, beginning-of-block markers (like { in Java), and end-of-block markers (}). Assume the language we're compiling for has the same scope rules as Java.



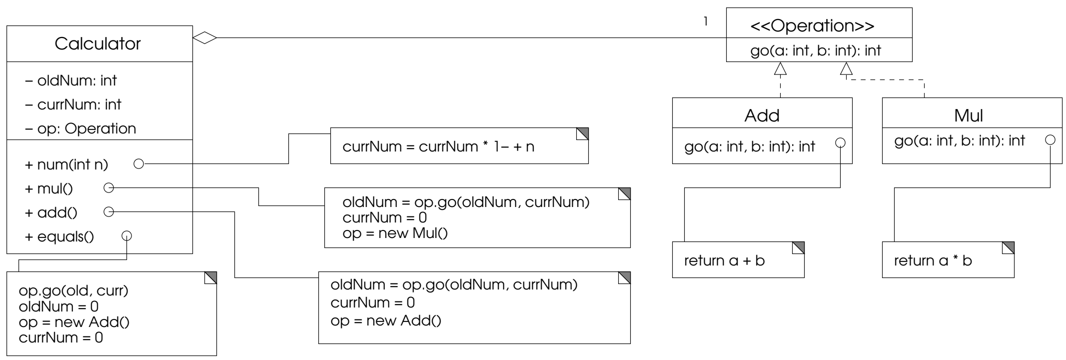
a. Use a UML diagram to design a class (and supporting classes, if necessary) implementing the following interface

```
public interface SymbTab {
   void enterScope();   // indicate we are entering a new block
   void associateVariable(String var, String type);  // indicate var has given type
   boolean isDeclared(String var);      // is this variable declared an in scope?
   String findType(String var);  // look up the type of a given variable
   void exitScope();   // leave the current block, get rid of symbols declared in that block
}
```

Also, explain your design in two or three sentences; identify what patterns and data structures, if any, you use or make variations on. You may use classes from the Java API. (8 points)

b. Write the class you designed in part a. (27 points)

3. Suppose the following diagram describes code you are given for a calculator program (which works only on nonnegative integers).



Now suppose you want to write a calculator like this but with two more buttons: ModOn and ModOff. When the ModOn button is pressed, the calculator switches to modular arithmetic, using whatever number is on the screen as the modulus. When the ModOff button is pressed, the calculator changes to normal arithmetic. Use a UML disgram to design a set of classes to implement this, making minimal changes to the classes you are given. Explain your design in a few sentences, identifying what patterns (if any) you use. (12 points)

4. Write a C function `trim()` which takes a character array containing a (heap-allocated) string and returns a character array which contains the same string but is no larger than necessary, deallocating the original character array. For example, if `msg` referred to a character array which we suspected took up more memory than necessary, we could use this function as

```
msg = trim(msg);
```

Note that it is `trim()`'s responsibility to deallocate the old array. (15 points)

```
char* trim(char* str)
{
```

5. Recall that a hash table implements a mapping or association using an array. For a given key, its index into the array is determined by finding its hash. In case more than one key is hashed to the same index, each array position actually contains a linked list of key-value pairs.

A hash *set* is similar except that the nodes in the lists contain only single items instead of pairs. Write a C struct (or structs) to implement a hash set of strings, and write the following C functions to operate on it.

```
HashSet* newHashSet(int arraySize);
void add(HashSet* set, char* item);
void remove(HashSet* set, char* item);
int contains(HashSet* set, char* item);
```

Note `newHashSet()` lets the caller specify the size of the array. Use the string's address mod the size of the array as a hash function, and also use their addresses for comparison (so, this would be like using == in Java, not `.equals()`). (22 points)