CS 340 Homework 4

Due Wednesday November 1
Homework 4 Part 1: Symbol Table

• Implement the symbol table class shown on the following slides.
• The implementation must use a separate chaining hash table
• You can choose your own hash function.
Symbol Table

import java.util.*;
public class SymbolTable {

    private class Node {
        // a node used to build linked lists
        // do not use the Java LinkedList class
        private String key;
        private Object data;
        private Node next;
        private Node(String k, Object d, Node x) {
            key = k;
            data = d;
            next = x;
        }
    }

    private Node table[];
    public SymbolTable(int s) {
        // s is the size of the table. You do not have to handle resizing the table
    }
}
private int hash(String k) {
    //return the hash function value for k
}

public boolean insert(String k) {
    //if k is not in the table create a new entry (with a null data value) and return true
    //if k is in the table return false
}
public boolean find(String k) {
    //return true if k is in the table otherwise return false
}

public Object getData(String k) {
    //if k is in the table return the data (which could be null) associated with k
    //if k is not in the table return null
}

public void setValue(String k, Object d) {
    //PRE: k is in the table
    //make d the data value associated with k
}
Symbol Table

public class STIterator implements Iterator<String> {
// An iterator that iterates through the keys in the table
    public STIterator() {
    }

    public boolean hasNext() {
    }

    public String next() {
// PRE: hasNext()
// The format of the string should be key:data where key is a key in the
// symbol table and data is the string representation of the data associated
// with the key
    }

    public void remove() {
// optional method not implemented
    }
}

Symbol Table

down

public boolean remove(String k) {
    //if k is in the table, return the entry for k and return true
    //if k is not in the table, return false

}

down

down

public Iterator<String> iterator() {
    //return a new STIterator object
}

down
public static void main(String args[]) {
    // code to test SymbolTable

}
Homework 4 Part 2: Expression Tree

• Implement and test the ExpressionTree class shown on the following slides.
• The constructor can assume the expression passed as a parameter is syntactically correct.
• Tokens (operands and operators) are separated by blanks
• evaluate returns the integer value of the expression tree. You will use a symbol table to lookup the current value of variables
• All calculations will be done with integer arithmetic
Expression operators

• The operators in precedence order are
  – ! (unary minus)
  – ^ (exponentiation)
  – *, /, %
  – +,-

• Exponentiation and unary minus are right associative
• All other operators are left associative
• Expressions can use parentheses to change the order of expression evaluation
public class ExpressionTree {

    private class Node {
        private Node left;
        private String data;
        private Node right;

        private Node(Node l, String d, Node r) {
            left = l;
            data = d;
            right = r;
        }
    }

    private Node root;
}
public ExpressionTree(String exp) {
    //PRE: exp is a legal infix expression
    //Build an expression tree from the expression exp
}
public int evaluate(SymbolTable t) {
    //return the int value of the expression tree
    //t is used to lookup values of variables
    return evaluate(t, root);
}

private int evaluate(SymbolTable t, Node r) {
    //return the int value of the expression tree with root r
    //t is used to lookup values of variables
    return evaluate(t, root);
}
ExpressionTree.java

public String toPostfix() {
    //return the postfix representation of the expression tree
    return toPostfix(root);
}

private String toPostfix(Node r) {
    //return the postfix representation of the tree with root r
}

public String toInfix() {
    //return the fully parenthesized infix representation of the expression tree
    return toInfix(root);
}

private String toInfix(Node r) {
    //return the fully parenthesized infix representation of the tree with root r
}
ExpressionTree.java

```java
public static void main(String args[]) throws IOException {
    // used to test expression tree
}
```
Homework 4 Part 3

• Implement a program that uses Symbol Table and Expression Tree to processes a file of assignment statements. The file contains one assignment statement per line. The syntax of an assignment statement is as follows

  Variable = Expression

• A variable is a letter followed by one or more letters or digits

• An expression is an infix arithmetic expression using the same operators discussed in part two of homework 4. An expression consists of integers, variables and operators
Homework 4 Part 3

• All tokens (integers, variables and operators) are separated by blanks
• You can assume that all the assignment statements are syntactically correct
• Use a symbol table from part 1 to lookup and store values of variables. A variable that has not been initialize is assumed to have a value of 0.
• After all the statements in the input file have been processed the values of all variables should be printed to standard output. (Use an STIterator)
• The name of the input file will be given as a command line argument.
Homework 4 Part 3

• Example input file

\[ x = 10 \]
\[ x^{Squared} = x^2 \]
\[ w = y + 3 \times x \]
\[ y = 2 \]
\[ z = 2 \times (x + y)^{(x - 8)} \]
Homework 4 Part 3

• Example output (the order of the final variable values could vary based on the hash function)

Final Variable Values
z 288
x 10
w 30
xSquared 100
y 2
Homework 4 Submission

• Submit the following files
  – SymbolTable.java
  – ExpressionTree.java
  – Assignments.java (the class that processes the assignments)
    • This file includes the main used to start the program
  – A test file of assignment statements that work for your program