Craig Dazey

CSE465 Final Project

Table of Contents

DataFile

I chose to do the data file problem in Java and Python. Overall, I think these were both good choices. This problem wasn't hard, long, or challenging by any means, but it really made me begin to appreciate how nice some of the features of these two languages are. Something that you have to do with this problem is split the list of scores up, and separate them into small subsets of the first list. In python this is extremely easy because there is array splicing which allows you to effectively take a substring of an array. Initially I was looking for a way to do this in java too, but I instead stumbled upon Streams. They are new in Java 8 and allow Java to act sort of as a functional language. In this way I was able to take subLists and use the ideas of map/filter/reduce to change whole lists from strings to ints in one line of code, as well as get the averages in one line of code. All of these features drastically increase the writeability of the code.

The development time was also significantly shorter for the python code. This is the kind of program that python was made to do. I didn't feel like I needed to make any classes, and the ability to just kind of fling different types around is incredibly freeing. The brevity of the code makes it extremely enjoyable to code in. For example, you can make a dictionary in python in less than 10 characters. This is opposed to Java where to make a HashMap I have to write a short biography. Also, I felt that making a class to hold the student data would be a good idea when doing it in Java. Because of things like Java's built in toString() method, and because of how wordy (try/catch, imports, etc..) java code can get, I thought it was a good idea to abstract the student info away from the main method. I didn't feel this way with python, most likely because you can do so much with so little code. I feel as though both languages were suitable for this project, but Python would definitely be my choice if I had to choose one. If the project were expanded even a little though, I would probably have to side with Java.

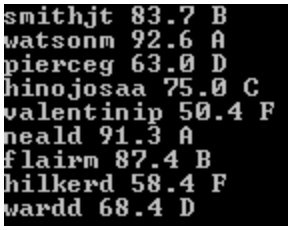
Because python is dynamically typed, the bigger the program gets, the harder it is to keep track of everything. In java, however, the static type checking at compile time will instantly let you know if you've messed up.

Statement of Correctness

All requirements were completed in all languages.

Output & Testing

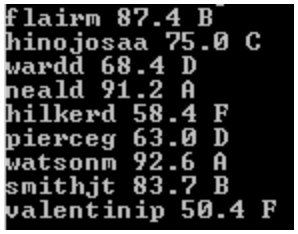
Java



Java - to Compile

javac DataFile.java Student.java
java DataFile grades.txt

Python



Python - to Compile

python Datafile.py

[Source Code Python]
[Source Code Java]
[Table of Contents]

NHL

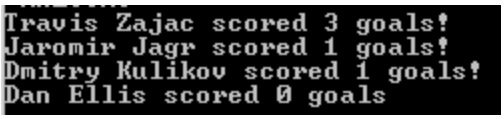
I chose to do the NHL program in Python and Java. In terms of language features, I believe Python definitely wins out. Opening and iterating through a file can be done in two lines of code, in less than 30 characters. In java you have to instantiate multiple objects and craft a semi-intricate loop to read through the file. Features like this severely hinder both the readability and writability of java code. It took me around 20 minutes more to write the java code just because it's such a verbose language, even though this is such a simple problem. Sure the python doesn't have any way to catch errors, but I'm not building a critical system, I'm just trying to parse a file for fantasy hockey. And when looking at the code now, I could still tell you exactly what the python code is doing on each line, while I would actually have to think to explain why I made some of the choices I did in the Java code. Overall, I don't think I would ever even consider doing a problem like this in Java if it wasn't required.

Statement of Correctness

All requirements were completed in all languages.

Output & Testing

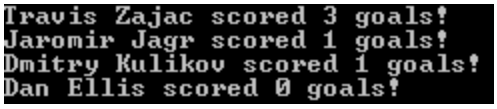
Java - Output



Java - to Compile

javac FantasyHockey.java
java FantasyHockey NHL.txt

Python - Output



Python - to Compile

python FantasyHockey.py

Source Code Python
Source Code Java
Table of Contents

Puzzle

I did the puzzle code in prolog and python. I think a problem like this is exactly what the prolog languages is meant to solve. The idea behind this puzzle is you need to find all the possible four way combinations you can make out of a list, and return true if one of these combinations has four parts that come to an equal sum. This is how prolog figures everything out to begin with. If tries every combination it can conceive to see if it can find something that satisfies its rules. The hardest part of the prolog code was trying to figure out how to write more clever code, it was never a question of how I was going to solve it. In python this wasn't really the case. I actually had to sit down and figure out an algorithm which required a little bit more of manual labor. And then implementing it took even longer. I finished the prolog code in around 30 seconds, with a grand total of 4 lines of code. In the python I had three for statements, totaling in at around 15 lines of code, and about 10-20 minutes of implementation time. This could have been drastically higher had I chose to do it in any other language. Because python has array splicing and the ability to call sum() on a list, I was able to check different partition sums very quickly. In a lot of other languages I would have had to make a few methods or have written a substantial amount of code. Both languages I think were pretty good choices for this problem. A year ago I would have done it in python without a second thought, but now having learned what prolog is, I can definitely see how it would apply to a problem like this.

Statement of Correctness

All requirements were completed in all languages.

Output & Testing

Prolog - Output

```
|    isPartitionable([1,2,3,3,2,1,1,2,3,3,1,2]).
true .

7 ?- isPartitionable([10,1,2,3,4,5,5,9,1]).
true .

8 ?- isPartitionable([1,1,1,1,4,3,1,1,3]).
true .

9 ?- isPartitionable([3,3,3,1,2,1]).
false.

10 ?- isPartitionable([1,1,1]).
false.

11 ?- isPartitionable([1,4,5,4,1,5,5]).
false.
```

Prolog - to Compile

Open the Puzzle.pl file in interactive mode
isPartitionable([1,2,3,3,2,1,1,2,3,3,1,2]).
isPartitionable([10,1,2,3,4,5,5,9,1]).
isPartitionable([1,1,1,1,4,3,1,1,3]).
isPartitionable([3,3,3,1,2,1]).
isPartitionable([1,1,1]).
isPartitionable([1,4,5,4,1,5,5]).

Python - Output

```
True
True
True
False
False
False
```

Python - to Compile

python Puzzle.py

[Source Code Python]

[Source Code Prolog]

[Table of Contents]

Expression

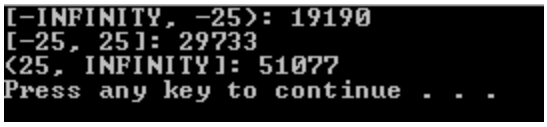
I did the expression problem in scheme and c++. When I was first thinking about what I was going to write for this section, I initially thought I was going to being completely on the scheme side. Given the exact problem we were, I still believe that scheme is the ideal choice. I think that because this problem really lends itself to be done recursively, and everything in scheme is done this way, it felt natural. The way lists are implemented in scheme made it a breeze to keep tacking on more operators and values. And then once the list was made it was practically already able to be evaluated. Just one method call that we didn't even have to implement and we could figure out the result of our expression trees. The only challenging thing to do in scheme was the adding to buckets and displaying the information. This was something I was definitely not used to and was pretty tricky at first. However, I think that the c++ version definitely has its merits. Because c++ is an OOP language, it lends itself to be expanded on. I think it would be hard to continue to add features to the scheme program. Really all we can do is make trees and evaluate them. In c++ we could expand on the trees in many different ways. We could have the nodes hold more data, or we could adjust how the trees are balanced in any way we want. The major downfall of the c++ code is because of its writeability. In c++ it's a constant battle to avoid leaking memory and deal with pointers. These become less of an issue though as you become more experiences. Overall I think that both of these languages we very capable of solving the problem at hand.

Statement of Correctness

All requirements were completed in all languages.

Output & Testing

C++ - Output



C++- to Compile

Visual Studio solution is attached

Scheme- Output

```
Number less than -25: 36296
Number between -25 && 25: 34139
Number greater than 25: 29565
```

Scheme - to Compile

Open the interactive thing
(load “Expression.scm”)
(findstats 1000)

[Source Code C++]
[Source Code Scheme]
[Table of Contents]

Z+-

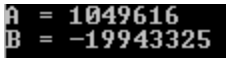
For this problem I used python and java. Overall, the process was almost identical with both. The python did have some quality of life improvements, mainly keeping me from having to type too much. This wasn't a huge factor though. In java I opted to do make a class to help with the parsing and storing of data. This was really from a readability standpoint as the main would be too cluttered to decipher what was happening if I left it whole. Because the problem was pretty tough to do in both languages, I really started to appreciate how much easier it was to write python code. All of the shortcuts you can take make it a whole heck of a lot more enjoyable. I think the greatest example of this is with the method I called buildForStatements(). In Java I had to create three different objects and loop through one of the objects and add+append stuff everywhere. Python, fortunately, has list comprehensions. By using a list comprehension I was able to condense 13 lines of code into 1. If I had to do it again, I would probably choose python just for the sake of my fingers and all the built in functions. The idea of "in" and the ease at which you can write for statements are examples of what makes python so writeable, and therefore more enjoyable to write in.

Statement of Correctness

All requirements were completed in all languages.

Output & Testing

Java - Output (Test 5)



Java - to Compile

javac InterpTest.java ZInterpreter.java
java InterpTest prog1.zpm
java InterpTest prog2.zpm
java InterpTest prog3.zpm

java InterpTest prog4.zpm
java InterpTest prog5.zpm
java InterpTest prog6.zpm

Python - Output

```
A = 1049616
B = -19943325
```

Python - to Compile

python ZInterpreter.py

[Source Code Python]
[Source Code Java]
[Table of Contents]

Appendix

DataFile

Python DataFile

```
__author__ = 'Craig'

from tkinter.filedialog import askopenfilename
from statistics import mean

def createFileOptions():
    options = {}
    options['defaultextension'] = '.txt'
    options['filetypes'] = [('all files', '.*'), ('text files', '.txt')]
    options['title'] = 'Please open a file with the grades you wish to calculate'
    return options

def getFileName():
    options = createFileOptions()
    name = askopenfilename(**options)
    return name

fileName = getFileName()
file = open(fileName)

gradeBook = {}

for line in file:
    parts = line.split()

    #change number strings to ints in list
    parts = [int(parts[i]) if parts[i].isdigit() else parts[i] for i in range(8)]

    name = parts[0]
    hw = float("{0:.1f}".format(mean(parts[1:4])))
```

```
    quiz = float("{0:.1f}".format(mean(parts[4:-1])))
    exam = float("{0:.1f}".format(parts[-1]))

    finalScore = float("{0:.1f}".format((hw * .35) + (quiz * .2) + (exam * .45)))

    if finalScore >= 90:
        letterGrade = 'A'
    elif finalScore >= 80:
        letterGrade = 'B'
    elif finalScore >= 70:
        letterGrade = 'C'
    elif finalScore >= 60:
        letterGrade = 'D'
    else:
        letterGrade = 'F'

    gradeBook[name] = {'hw': hw, 'quiz': quiz, 'exam': exam, 'finalScore': finalScore,
'letterGrade': letterGrade}

for name in gradeBook:
    print('{} {} {}'.format(name, gradeBook[name]['finalScore'],
gradeBook[name]['letterGrade']))
```

Java DataFile

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class DataFile {
    static Student makeStudent(String stats)    {
```

```
    List<String> parts = Arrays.asList(stats.split(" "));

    String name = parts.get(0);
    List<Integer> homework = parts.subList(1, 4).stream()
            .map(Integer::valueOf).collect(Collectors.toList());
    List<Integer> quiz = parts.subList(4, parts.size() - 1).stream()
            .map(Integer::valueOf).collect(Collectors.toList());
    int exam = Integer.parseInt(parts.get(parts.size() - 1));

    return new Student(name, homework, quiz, exam);
}

public static void main(String[] args)  {
    String fileName = args[0];
    BufferedReader input = null;
    ArrayList<Student> students = new ArrayList<Student>();
    String line;

    try {
        input = new BufferedReader(new FileReader(fileName));
        while((line = input.readLine()) != null)    {
            students.add(makeStudent(line));
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        System.out.println("Failed to open the file");
    } catch (IOException e) {
        e.printStackTrace();
        System.out.println("Error reading line from file");
    } finally {
        try {
            if(input != null) input.close();
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("Error closing bufferedReader");
        }
```

```
        }

        for(Student s : students)   {
            System.out.println(s.toString());
        }
    }
}

import java.util.ArrayList;
import java.util.List;

public class Student {
    private String name;
    private List<Integer> homeworkGrades;
    private List<Integer> quizGrades;
    private double homeworkScore;
    private double quizScore;
    private double examScore;

    public Student(String name, List<Integer> homework, List<Integer> quiz, int exam)
{
        this.name = name;
        homeworkGrades = homework;
        quizGrades = quiz;

        homeworkScore = homeworkAverage();
        quizScore = quizAverage();
        examScore = exam;
    }

    double homeworkAverage()    {
        return homeworkGrades.stream().mapToInt(i ->
i).average().orElseThrow(IllegalArgumentException::new);
    }
```

```
    double quizAverage()    {
        return quizGrades.stream().mapToInt(i ->
i).average().orElseThrow(IllegalArgumentException::new);
    }

    public double getGrade()    {
        return homeworkScore * .35 + quizScore * .20 + examScore * .45;
    }

    public char getLetterGrade()    {
        double grade = getGrade();
        if(grade > 90)
            return 'A';
        else if(grade > 80)
            return 'B';
        else if(grade > 70)
            return 'C';
        else if(grade > 60)
            return 'D';
        else
            return 'F';
    }

    public String getName() {
        return this.name;
    }

    public String toString()    {
        return String.format("%s %.1f %c", this.name, this.getGrade(),
this.getLetterGrade());
    }
}
```

NHL

Python NHL

```
__author__ = 'Craig'

from tkinter.filedialog import askopenfilename

def createFileOptions():
    options = {}
    options['defaultextension'] = '.txt'
    options['filetypes'] = [('all files', '.*'), ('text files', '.txt')]
    options['title'] = 'Please open a file with the NHL information you wish to
process'
    return options

def getFileName():
    options = createFileOptions()
    name = askopenfilename(**options)
    return name

'''Returns a string in the form of: NAME scored INT goals!'''
def getNumGoals(goals, name):
    if not name in goals:
        return '{} scored {} goals!'.format(name, 0)
    else:
        return '{} scored {} goals!'.format(name, goals[name])

fileName = getFileName()
file = open(fileName)

goals = {}

for line in file:
    if "goal scored" in line.lower():
        parts = line.split()
        indexOfName = parts.index('by') + 1
```

```
        firstName = parts[indexOfName]
        lastName = parts[indexOfName + 1].split('(')[0]
        name = '{} {}'.format(firstName, lastName)

        if not name in goals:
            goals[name] = 1
        else:
            goals[name] += 1

print(getNumGoals(goals, 'Travis Zajac'))
print(getNumGoals(goals, 'Jaromir Jagr'))
print(getNumGoals(goals, 'Dmitry Kulikov'))
print(getNumGoals(goals, 'Dan Ellis'))
```

Java NHL

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class FantasyHockey {

    static void printPlayerScore(String name, HashMap<String, Integer> goals)   {
        if(!goals.containsKey(name))    {
            System.out.println(String.format("%s scored 0 goals", name));
            return;
        }

        System.out.println(String.format("%s scored %d goals!", name,
```

```
goals.get(name)));

    }
    public static void main(String[] args)  {
        final String SEARCH = "goal scored";

        String fileName = args[0];
        BufferedReader input = null;
        String line;
        HashMap<String, Integer> playerGoals = new HashMap<String, Integer>();

        try {
            input = new BufferedReader(new FileReader(fileName));
            while((line = input.readLine()) != null)    {
                if(line.toLowerCase().contains(SEARCH)) {
                    List<String> parts = Arrays.asList(line.split(" "));
                    for(int i = 0; i < parts.size() - 1; i++)   {
                        if (parts.get(i).equals("by"))  {
                            String firstName = parts.get(i + 1);
                            String lastName = parts.get(i + 2);
                            lastName = Arrays.asList(lastName.split("\\(")).get(0);

                            String name = firstName + " " + lastName;

                            if(playerGoals.containsKey(name))   {
                                playerGoals.put(name, playerGoals.get(name) + 1);
                                break;
                            } else  {
                                playerGoals.put(name, 1);
                                break;
                            }
                        }
                    }
                }
            }
            printPlayerScore("Travis Zajac", playerGoals);
```

```
            printPlayerScore("Jaromir Jagr", playerGoals);
            printPlayerScore("Dmitry Kulikov", playerGoals);
            printPlayerScore("Dan Ellis", playerGoals);

        } catch (FileNotFoundException e) {
            e.printStackTrace();
            System.out.println("Failed to open the file");
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("Error reading line from file");
        } finally {
            try {
                if(input != null) input.close();
            } catch (IOException e) {
                e.printStackTrace();
                System.out.println("Error closing bufferedReader");
            }
        }
    }
}
```

Puzzle

Python Puzzle

```
__author__ = 'Craig'


def isPartitionable(nums):
    ret = False
    for i in range(len(nums)):
        part1 = nums[i:]
        part2 = nums[:i]
        if(sum(part1) == sum(part2)):
            for j in range(len(part1)):
                if(sum(part1[j:]) == sum(part1[:j])):
                    for k in range(len(part2)):
                        if(sum(part2[k:]) == sum(part2[:k])):
                            ret = True
    return ret



print(isPartitionable([1,2,3,3,2,1,1,2,3,3,1,2]))
print(isPartitionable([10,1,2,3,4,5,5,9,1]))
print(isPartitionable([1,1,1,1,4,3,1,1,3]))
print(isPartitionable([3,3,3,1,2,1]))
print(isPartitionable([1,1,1]))
print(isPartitionable([1,4,5,4,1,5,5]))
```

Prolog Puzzle

```
sum([], 0).
sum([H|T], Sum) :- sum(T, X), Sum is H + X.

partitionable(L) :- append(X, Y, L), sum(X, P), sum(Y, P), !.
isPartitionable(L) :- append(X, Y, L), partitionable(X), partitionable(Y).
```

Expression

C++ Expression

```
using namespace std;

#include <iostream>
#include <string>
#include <vector>
#include <stdlib.h>
#include <time.h>
#include <sstream>
#include <math.h>

struct Node     {
    Node* left;
    Node* right;
    char op;
    bool isVar;
    int value;
    ~Node();
};

class Expression    {
public:
    Expression();
    Expression(string str);
    Expression(const Expression &other);
    Expression &operator = (const Expression &other);
    ~Expression();
    double evaluate(double x, bool y) const;
    static Expression randomExpression(int height);
    string toString() const;
    void mutate();
    int getHeight() const;
private:
    Node *root;
```

```
    Node* parseStatement(string str);
    int convertToInt(string str) const;
    bool isInteger(const string & s);
    int findIndexMiddleExp(string str);
    char getMiddleExp(string str);
    vector<string> getLeftAndRight(string str);
    string stripParens(string str);
    Node* copyHelper(const Node *node);
    int getHeightFromNode(Node* other) const;
    double evaluateFromNode(Node* node, double x, bool y) const;
    static Node* addRandomNode(Node* root, int count, int height);
    void changeNode(Node* node);
    int subTreeNodeCount(Node* node);
    string getStringSubTree(Node* node, string str) const;
    string nodeToString(Node* node) const;
};

#include "Expression.h"

Expression::Expression()    {
    root = new Node();
    root->value = 0;
    root->isVar = NULL;
    root->op = NULL;
    root->left = NULL;
    root->right = NULL;
}

Expression::Expression(string str)  {
    str = stripParens(str);
    root = parseStatement(str);
}

Expression::Expression(const Expression &other) {
    root = copyHelper(other.root);
}
```

```
Expression &Expression::operator = (const Expression &other)    {
    if (this != &other) {
        delete root;
        root = copyHelper(other.root);
    }
    return *this;
}

Expression::~Expression()   {
    delete root;
}

Node::~Node()   {
    delete left;
    delete right;
    left = nullptr;
    right = nullptr;
}

double Expression::evaluate(double x, bool y) const {
    return evaluateFromNode(root, x, y);
}

Expression Expression::randomExpression(int height) {
    Expression exp;
    exp.root = new Node();
    exp.root->isVar = false;
    exp.root->value = NULL;
    exp.root->left = NULL;
    exp.root->right = NULL;

    int chance = rand() % 4;
    switch (chance) {
        case 0:
            exp.root->op = '+';
```

```
            break;
        case 1:
            exp.root->op = '-';
            break;
        case 2:
            exp.root->op = '*';
            break;
        case 3:
            exp.root->op = '^';
            break;
    }

    vector<Node*> oldNodes;
    vector<Node*> newNodes;
    oldNodes.push_back(exp.root);
    int count = 0;
    while (count < height)  {
        for (Node* node : oldNodes) {
            if (node->value == NULL)    {
                node->left = addRandomNode(node, count, height);
                node->right = addRandomNode(node, count, height);
                newNodes.push_back(node->left);
                newNodes.push_back(node->right);
            }
        }
        oldNodes.clear();
        oldNodes = newNodes;
        newNodes.clear();
        count += 1;
    }

    return exp;
}

string Expression::toString() const {
    string ret = "";
```

```
    string str = getStringSubTree(root, ret);
    return str;
}

void Expression::mutate()   {
    Node* node = root;

    while (true)    {
        int nodeCount = subTreeNodeCount(node);
        int val = rand() % nodeCount;
        if (val == 0)   {
            changeNode(node);
            break;
        }
        else if (val <= subTreeNodeCount(node->left)){
            node = node->left;
        }
        else{
            node = node->right;
        }
    }
}

int Expression::getHeight() const   {
    return getHeightFromNode(root) - 1; //don't include the root node
}

//============================HELPER FUNCTIONS============================

Node* Expression::parseStatement(string str)    {
    if (getMiddleExp(str) == '+' || getMiddleExp(str) == '-' || getMiddleExp(str) ==
'*' || getMiddleExp(str) == '^')   {
        vector<string> parts = getLeftAndRight(str);
        Node* newOpNode = new Node();
        newOpNode->op = getMiddleExp(str);
        newOpNode->isVar = false;
```

```
        newOpNode->value = NULL;
        newOpNode->left = parseStatement(stripParens(parts[0]));
        newOpNode->right = parseStatement(stripParens(parts[1]));
        return newOpNode;
    }
    if (isdigit(getMiddleExp(str))) {
        Node* newValNode = new Node();
        newValNode->op = NULL;
        newValNode->isVar = false;
        char c = getMiddleExp(str);
        newValNode->value = c - '0';
        newValNode->left = NULL;
        newValNode->right = NULL;
        return newValNode;
    }
    if (getMiddleExp(str) == 'x')   {
        Node* newVarNode = new Node();
        newVarNode->op = NULL;
        newVarNode->isVar = true;
        newVarNode->value = NULL;
        newVarNode->left = NULL;
        newVarNode->right = NULL;
        return newVarNode;
    }
}

int Expression::convertToInt(string str) const  {
    std::istringstream converter(str);
    int ret;
    converter >> ret;
    return ret;
}

bool Expression::isInteger(const string & s)
{
    if (s.empty() || ((!isdigit(s[0])) && (s[0] != '-') && (s[0] != '+')))
```

```
        return false;
    char* p;
    strtol(s.c_str(), &p, 10);

    return (*p == 0);
}

int Expression::findIndexMiddleExp(string str)  {
    int left = 0;
    int right = 0;

    if (str.find('(') == -1)    {
        return str.size() / 2;
    }

    for (int i = 0; i < str.size(); i++)    {
        if (str[i] == '(')  {
            left += 1;
        }
        if (str[i] == ')')  {
            right += 1;
        }
        if (left == right)  {
            if (i != str.size() - 1)    {
                return i + 1;
            }
            else    {
                return i / 2;
            }
        }
    }
}

char Expression::getMiddleExp(string str)   {
    int i = findIndexMiddleExp(str);
    return str[i];
```

```
}

vector<string> Expression::getLeftAndRight(string str)  {
    std::vector<std::string> parts;
    parts.push_back(str.substr(0, findIndexMiddleExp(str)));
    parts.push_back(str.substr(findIndexMiddleExp(str) + 1, string::npos));
    return parts;
}

string Expression::stripParens(string str)  {
    if (str.find('(') != -1)    {
        str = str.substr(1, str.length() - 2);
        return str;
    }
    return str;
}

Node* Expression::copyHelper(const Node *other) {
    if (other == NULL)  {
        return NULL;
    }

    Node *newnode = new Node();
    newnode->op = other->op;
    newnode->isVar = other->isVar;
    newnode->value = other->value;
    newnode->left = copyHelper(other->left);
    newnode->right = copyHelper(other->right);
    return newnode;
}

int Expression::getHeightFromNode(Node* node)   const{
    int heightLeft = 0;
    int heightRight = 0;
    if (node->left != NULL)
        heightLeft = getHeightFromNode(node->left);
```

```
    if (node->right != NULL)
        heightRight = getHeightFromNode(node->right);
    if (heightLeft > heightRight){
        return heightLeft + 1;
    }
    else{
        return heightRight + 1;
    }
}

double Expression::evaluateFromNode(Node* node, double x, bool y) const {
    if (node->left != NULL && node->right != NULL)  {
        double leftVal = evaluateFromNode(node->left, x, y);
        double rightVal = evaluateFromNode(node->right, x, y);
        char op = node->op;
        switch (op) {
        case '+':
            return leftVal + rightVal;
            break;
        case '-':
            return leftVal - rightVal;
            break;
        case '*':
            return leftVal * rightVal;
            break;
        case '^':
            return pow(leftVal, rightVal);
            break;
        }
        double val = leftVal + rightVal;
        return val;
    }
    else{
        if(y)
            return rand() % 10;
        else if (node->isVar)
```

```
            return x;
        else
            return node->value;
    }
}

Node* Expression::addRandomNode(Node* node, int count, int height){
    if (count != height - 1)    {
        Node* newNode = new Node();
        int chance = rand() % 4;
        switch (chance) {
        case 0:
            newNode->op = '+';
            break;
        case 1:
            newNode->op = '-';
            break;
        case 2:
            newNode->op = '*';
            break;
        case 3:
            newNode->op = '^';
            break;
        }
        newNode->isVar = false;
        newNode->value = NULL;
        newNode->left = NULL;
        newNode->right = NULL;
        return newNode;
    }
    else {
        if (rand() % 2 == 0)    {
            Node* newNode = new Node();
            newNode->isVar = true;
            newNode->op = NULL;
            newNode->value = NULL;
```

```
            newNode->left = NULL;
            newNode->right = NULL;
            return newNode;
        }
        else {
            Node* newNode = new Node();
            newNode->value = rand() % 10;
            newNode->isVar = false;
            newNode->op = NULL;
            newNode->left = NULL;
            newNode->right = NULL;
            return newNode;
        }
    }
}

void Expression::changeNode(Node* node) {
    if (node->isVar != NULL && node->isVar == true) {
        node->isVar = false;
        node->value = rand() % 10;
    }
    else if (node->op != NULL){
        int chance = rand() % 4;
        switch (chance) {
        case 0:
            node->op = '+';
            break;
        case 1:
            node->op = '-';
            break;
        case 2:
            node->op = '*';
            break;
        case 3:
            node->op = '^';
            break;
```

```
        }
    }
    else{
        int chance = rand() % 10;
        node->value = chance;
    }
}

int Expression::subTreeNodeCount(Node* node)    {
    if (node == NULL)   {
        return 0;
    }
    else {
        return 1 + subTreeNodeCount(node->left) + subTreeNodeCount(node->right);
    }
}

string Expression::getStringSubTree(Node* node, string str) const{
        if (node->op == NULL)   {
            return nodeToString(node);
        }
        else {
            string temp = "";
            temp += "(";
            temp += getStringSubTree(node->left, str);
            temp += node->op;
            temp += getStringSubTree(node->right, str);
            temp += ")";
            return temp;
        }
}

string Expression::nodeToString(Node* node) const{
    string str = "";
    if (node->op == NULL && !node->isVar)   {
        str += "(";
```

```
        char c = node->value + '0';
        str += c;
        str += ")";
    }
    else if (node->isVar){
        str += "(";
        str += "x";
        str += ")";
    }
    return str;
}

/*
* Craig Dazey
*/

#include <iostream>
#include "Expression.h"
using namespace std;

vector<int> makeBuckets(int trials, int height) {
    vector<int> buckets(3);
    Expression exp;

    for(int i = 0; i < trials; i++) {
        exp = Expression::randomExpression(height);
        double result = exp.evaluate(0, true);

        if(result < -25)
            buckets[0] += 1;
        else if(result > 25)
            buckets[2] += 1;
        else
            buckets[1] += 1;
    }
```

```
    return buckets;
}

int main()  {
    srand(time(0));

    vector<int> buckets = makeBuckets(100000, 3);

    cout << "[-INFINITY, -25): " << buckets[0] << endl;
    cout << "[-25, 25]: " << buckets[1] << endl;
    cout << "(25, INFINITY]: " << buckets[2] << endl;

    return 0;
}
```

Scheme Expression

```
(define (randop)
  (define x (random 3))
  (cond
   ((= x 0) '+)
   ((= x 1) '*)
   ((= x 2) '-))
)

(define (createTree height)
  (cond
    ((= height 0) (list (randop) (random 10) (random 10)))
    ((list (randop) (createTree (- height 1)) (createTree(- height 1))))
  )
)

(define (testExpTree lst)
    (cond
        ((= (car lst) 0) lst)
        ((< (eval (createTree 3) user-initial-environment) (- 25)) (testExpTree(list (-
```

```
(car lst) 1) (+ 1 (cadr lst)) (caddr lst) (cadddr lst))))
        ((> (eval (createTree 3) user-initial-environment) 25) (testExpTree(list (-
(car lst) 1) (cadr lst) (caddr lst) (+ 1 (cadddr lst)))))
        (ELSE (testExpTree(list (- (car lst) 1) (cadr lst) (+ 1 (caddr lst)) (cadddr
lst))))
     )
)

(define (findstats trials)
    (define lst (testExpTree (list trials 0 0 0)))
    (display "Number less than -25: ")
    (display (cadr lst))
    (NEWLINE)
    (display "Number between -25 && 25: ")
    (display (caddr lst))
    (NEWLINE)
    (display "Number greater than 25: ")
    (display (cadddr lst))
)
```

Z+-

Python Z+-

```
#Craig Dazey

from tkinter.filedialog import askopenfilename

var = {}

def createFileOptions():
    options = {}
    options['defaultextension'] = '.txt'
    options['filetypes'] = [('all files', '.*'), ('text files', '.txt')]
    options['initialfile'] = r'C:\Users\damonde\Desktop\CSE465\Homework1\prog6.zpm'
    options['title'] = 'Open source file'
    return options

def getFileName():
    options = createFileOptions()
    name = askopenfilename(**options)
    return name

def parseFile(fileName):
    try:
        file = open(fileName, 'r')
        for line in file:
            if line.strip():
                parseLine(line)
    except Exception:
        print(traceback.format_exc())

def parseLine(line):
    parts = line.split()
    if parts[0] == 'DEF':
        var[parts[1]] = 0
    elif parts[0] == 'FOR':
```

```
            statements = buildForStatements(parts[2:])
            count = int(parts[1])
            for i in range(count):
                for s in statements:
                    doOp(s)
        elif parts[0] in var:
            if parts[1] == '=':
                if parts[2] in var:
                    var[parts[0]] = var[parts[2]]
                else:
                    var[parts[0]] = int(parts[2])
            else:
                doOp(parts)

def buildForStatements(parts):
    parts = [s.split() for s in (" ".join(parts)).split(';')]
    return parts[:-1]

def doOp(parts):
    if '+' in parts[1]:
        if parts[2] in var:
            var[parts[0]] = var[parts[0]] + var[parts[2]]
        else:
            var[parts[0]] = var[parts[0]] + int(parts[2])
    if '-' in parts[1]:
        if parts[2] in var:
            var[parts[0]] = var[parts[0]] - var[parts[2]]
        else:
            var[parts[0]] = var[parts[0]] - int(parts[2])
    if '*' in parts[1]:
        if parts[2] in var:
            var[parts[0]] = var[parts[0]] * var[parts[2]]
        else:
            var[parts[0]] = var[parts[0]] * int(parts[2])
```

```
def printVars():
    for key in sorted(var.keys()):
        print(key + ' = ' + str(var[key]))


if __name__ == '__main__':
    fileName = getFileName()
    parseFile(fileName)
    printVars()
```

Java Z+-

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map.Entry;

/**
 * Created by Craig on 1/31/2015.
 */

public class ZInterpreter {

    private HashMap<String, Integer> vars;

    public ZInterpreter()   {
        vars = new HashMap();
    }

    public void parseLine(String line) {
        String[] parts = line.split(" ");

        if (parts[0].contains("DEF")) {       //DEF a ;
            vars.put(parts[1], 0);
        } else if (parts[0].contains("FOR")) {
            ArrayList<String> statements = buildForStatements(parts);
```

```
            int count = Integer.parseInt(parts[1]);
            for(int i = 0; i < count; i++)  {
                for(String s: statements)   {
                    String[] statement = s.split(" ");
                    doOp(statement);
                }
            }

        } else if (vars.containsKey(parts[0])) {       //starts with letter in hashmap
            if (parts[1].equals("=")) {      // a =
                if (vars.containsKey(parts[2])) {       //a = another letter in hashmap
                    vars.put(parts[0], vars.get(parts[2]));
                } else {        // a = number
                    vars.put(parts[0], Integer.parseInt(parts[2]));
                }
            } else {
                doOp(parts);
            }
        }
    }

    public void doOp(String[] parts)  {     //pass this guy something like a = 3
        if(parts[1].contains("+"))  {
            if (vars.containsKey(parts[2])) {
                vars.put(parts[0], vars.get(parts[2]) + vars.get(parts[0]));
            } else {
                vars.put(parts[0], Integer.parseInt(parts[2]) + vars.get(parts[0]));
            }
        }
        if(parts[1].contains("-"))  {
            if (vars.containsKey(parts[2])) {
                vars.put(parts[0], vars.get(parts[0]) - vars.get(parts[2]));
            } else {
                vars.put(parts[0], vars.get(parts[0]) - Integer.parseInt(parts[2]));
            }
        }
```

```
    if(parts[1].contains("*")) {
        if (vars.containsKey(parts[2])) {
            vars.put(parts[0], vars.get(parts[2]) * vars.get(parts[0]));
        } else {
            vars.put(parts[0], Integer.parseInt(parts[2]) * vars.get(parts[0]));
        }
    }
}

public ArrayList<String> buildForStatements(String[] parts)  {
    ArrayList<String> words = new ArrayList<String>(Arrays.asList(parts));
    ArrayList<String> ret = new ArrayList<String>();

    StringBuilder sb = new StringBuilder();
    for(int i = 2; !words.get(i).equals("ENDFOR"); i++) {
        if(words.get(i).equals(";")) {
            ret.add(sb.toString());         //need to split this
            sb.setLength(0);
        }
        else {
            sb.append(words.get(i) + " ");      //Definitely hacky, but it works
        }
    }

    return ret;
}

public void printVars() {
    for(Entry<String, Integer> entry: vars.entrySet()){
        System.out.println(entry.getKey() + " = " + entry.getValue());
    }
}

public void clearInterpreter()  {       //method to clear vars for time testing
    this.vars.clear();
```

```
    }
}

import java.io.*;

/**
 * Created by Craig on 1/31/2015.
 */

public class InterpTest {
    public static void main(String[] args) throws IOException {

        ZInterpreter interp = new ZInterpreter();
        BufferedReader br = new BufferedReader(new FileReader(args[0]));
        String line = null;
        while ((line = br.readLine()) != null) {
            interp.parseLine(line);
        }
        interp.printVars();
    }
}
```