CSE465 Comparative Programming Term Project

Nick Venenga

Contents

Mail Merge

C#

Analysis

C# was fairly well suited for creating an easy to use mail merge program. Especially with the help of Visual Studio, C# is a relatively easy language to write in and has well-developed documentation. In addition, the language has a fairly comprehensive standard library that made string processing a breeze. For instance, C# has a built-in method to split a string and store the parts directly into an array of strings without having to loop over command output or determine the number of parts created to manually instantiate an array of the correct size. In addition, C# string tools allow easy replacing of the placeholders in the template file without having to do any additional loops or character-by-character searching. Not only are the string tools easy to use, C# provides simple ways to open text files, iterate over the lines and quickly save information to new files. Overall, the program took almost no time to development (maybe 15 minutes including testing time) and was extremely easy to test and run on Windows (once again using Visual Studio). Even with little time spent using C#, it was easy to pick up the language. In regards to runtime performance, I think that is largely a moot-point for this sort of program. The most likely bottleneck for this program would be disk IO so any benefits of using a compiled language over an interpreted language are largely irrelevant. If running on a server or a system dedicated to mail merging, C# may present an edge over Python, but it also had a slightly longer development time. I feel out of the compiled languages I've used, C# was probably the easiest to do string manipulation in.

Correctness

The program functions are expected and will find the column named "ID" correctly and use it regardless of position.

Output

Template

```
<<COURSE>><<COURSE>>
(<<ID>>)
there will be other tests too.
```

Results

```
565565
(haygoodb)
there will be other tests too.
```

Code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CSE465
{
    class mm
    {
        public const char DELIM = '\t';
        public static int UID_Offset;
        public static string merge(String tmpl, string[] fields, string[]
replacements)
        {
            for (int i = 0; i < fields.Length; i++) {
                tmpl = tmpl.Replace("<<" + fields[i] + ">>", replacements[i]);
            }
            return tmpl;
        }
        public static void save(string filename, string text)
        {
            StreamWriter output = new StreamWriter(filename + ".txt");
            output.Write(text);
            output.Close();
        }
        static void Main(string[] args)
        {
            // Load the template
            StreamReader input = new StreamReader(args[args.Length-1]);
            string tmpl = input.ReadToEnd();
            input.Close();

            // Load the data file
            input = new StreamReader(args[args.Length-2]);

            //Get the fields
            string[] fields = input.ReadLine().Split(DELIM);
            UID_Offset = Array.IndexOf(fields, "ID");

            string line, temp;
            string[] data;
            while((line = input.ReadLine()) != null)
            {
                data = line.Split(DELIM);
                temp = merge(tmpl, fields, data);
                save(data[UID_Offset], temp);
            }
        }
    }
}
```

Python

Analysis

If C# was easy to write a mail merge program in, Python was child's play. Python's extensive built-in string tools made short work of reading a file and replacing strings in a template. The loose typing made storing the data easy as well. As usual, an interpreted language made writing a short script—especially one for processing text—effortless. I've previously used Python for process logging files and other text documents so there was no learning curve at all for the assignment. Even if I didn't know Python, though, there is a strong community online and it's very easy to find documentation and examples. One may argue that an interpreted language would have worse performance than a compiled one, but, as mentioned above, the primary bottleneck in this case is likely disk IO. In addition, it's common for mail merge programs to be run on the desktop by end-users creating newsletters or similar items where the computer has plenty of RAM and CPU and small performance differences aren't noticeable. As mentioned, I've previously used Python for processing text files and will continue to go to it in the future due to the easy of development and great portability (many *nix systems come with at least Python 2 if not Python 3 and all major platforms have Python binaries available). For portability, Python has a clear advantage over C# which only recently released a compiler for Linux systems.

Correctness

Program is fully functional.

Output

Template

```
<<COURSE>><<COURSE>>
(<<ID>>)
there will be other tests too.
```

Results

```
565565
(haygoodb)
there will be other tests too.
```

Code

```
#!/usr/bin/python3

import sys

delim = '\t' # Specify the data delimiting character
dataFilename = sys.argv[1]
tmplFilename = sys.argv[2]

# Replace everything inside << >> with corresponding data
def merge(template, data):
	for field in data:
		template = template.replace('<<' + field + '>>', data[field])
	return template

# Save text to a file
def save(text, filename):
	with open(filename, 'w') as file:
		file.write(text)

# Build template string from multiple lines
tmpl = ""
with open(tmplFilename, 'r') as tmplFile:
	tmpl = "".join(tmplFile.readlines()) # Just because it's Python--why not

with open(dataFilename, 'r') as dataFile:
	# Get field names
	fields = dataFile.readline().rstrip().split(delim)
	# Go through each data line
	for line in dataFile:
		# Remove new line character -> get each string
		rpl_strings = line.rstrip().split(delim)
		# Create a dictionary out of field names and data
		replacements = dict(zip(fields, rpl_strings))
		# Merge dictionary with template
		text = merge(tmpl, replacements)
		# Save results
		save(text, replacements['ID'] + '.txt')
```

Great Uncle/Aunt

Java

Analysis

It was fairly easy to write the program in Java. The biggest challenge was creating the tree data structure to hold the data. It seems like Java tries to have an answer to all problems and as a result has an absolute over-abundance of similar objects and interfaces. Part of the development time was spent investigating whether Java had built-in tree functionality for creating simple trees. I ended up creating a Node class modeled after an example I read on StackOverflow. I decided to add marriages as a sort of side-link on each node. I just went left-to-right when looking at the graphical example to decide which person to make the primary and which node to set as partner. To speed up development time, I skipped getters and setters in many cases, but they don't effect the underlying functionality—just maintainability. I had to consult OpenDatastructures for a refresher on searching and implemented a modified depth-first search that took into account the linked partner nodes. Overall I think development in Java was rather painless, but setting everything up correctly still took a decent amount of time—luckily Java provided a wealth of tools for creating the desired effect. In regards to runtime performance, the problem is so small there's no noticeable difference. There is overhead in launching the Java virtual machine but it is definitely less than the interpreters for Scheme and Prolog. It's hard to give an unbiased opinion on the learning curve involved with Java since that is the beginning language taught at Miami, and most people have a strong background using Java. Overall, I think Java was a good choice but it felt a little like rolling in a 500lb tool chest when only a single wrench was needed.

Correctness

The program produces the correct output.

Output

```
System.out.println(greatUncleAunt("male6", "female7"));
System.out.println(greatUncleAunt("male6", "male8"));
System.out.println(greatUncleAunt("female7", "male3"));
System.out.println(greatUncleAunt("female4", "female7"));
System.out.println(greatUncleAunt("female4", "male3"));
System.out.println(greatUncleAunt("male3", "male7"));
System.out.println(greatUncleAunt("male6", "female7"));
System.out.println(greatUncleAunt("female7", "male6"));
System.out.println(greatUncleAunt("male5", "female8"));
System.out.println(greatUncleAunt("male5", "male8"));
System.out.println(greatUncleAunt("male8", "female8"));
```

```
true
false
false
true
false
true
true
false
false
false
false
```

Code

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

/**
 * Comparative program great uncle/aunt problem
 * @author Nick Venenga et al (see comments in code for sources)
 */
public class uncle {
	public static Node<String> Tree; //easier to store the free up here so it's 
not getting passed all over the place

	public static void main(String args[]) {
		Tree = buildFamilyTree();

		//System.out.println(greatUncleAunt("", "", tree));
		System.out.println(greatUncleAunt("male6", "female7"));
		System.out.println(greatUncleAunt("male6", "male8"));
		System.out.println(greatUncleAunt("female7", "male3"));
		System.out.println(greatUncleAunt("female4", "female7"));
		System.out.println(greatUncleAunt("female4", "male3"));
		System.out.println(greatUncleAunt("male3", "male7"));
		System.out.println(greatUncleAunt("male6", "female7"));
		System.out.println(greatUncleAunt("female7", "male6"));
		System.out.println(greatUncleAunt("male5", "female8"));
		System.out.println(greatUncleAunt("male5", "male8"));
		System.out.println(greatUncleAunt("male8", "female8"));
	}

	/**
	 * True if different of two generations
	 * @param uncleAunt
	 * @param person
	 * @return whether it's a great uncle/aunt
	 */
	public static boolean greatUncleAunt(String uncleAunt, String person) {
		return Tree.findDepth(uncleAunt)+2 == Tree.findDepth(person);
	}

	/**
	 * Builds a family free using information about relations
	 * from homework
	 * @return root node of family tree
	 */
	public static Node<String> buildFamilyTree() {
		Node<String> root = new Node<String>("root");

		//Create an object for search male
		Node<String> male1 = new Node<String>("male1");
		Node<String> male2 = new Node<String>("male2");
		Node<String> male3 = new Node<String>("male3");
		Node<String> male4 = new Node<String>("male4");
		Node<String> male5 = new Node<String>("male5");
```

```
        Node<String> male6 = new Node<String>("male6");
        Node<String> male7 = new Node<String>("male7");
        Node<String> male8 = new Node<String>("male8");

        //Create an object for each female
        Node<String> female1 = new Node<String>("female1");
        Node<String> female2 = new Node<String>("female2");
        Node<String> female3 = new Node<String>("female3");
        Node<String> female4 = new Node<String>("female4");
        Node<String> female5 = new Node<String>("female5");
        Node<String> female6 = new Node<String>("female6");
        Node<String> female7 = new Node<String>("female7");
        Node<String> female8 = new Node<String>("female8");
        Node<String> female9 = new Node<String>("female9");
        Node<String> female10 = new Node<String>("female10");

        //Level 1
        root.addChild(male1);
        male1.setPartner(female1);

        //Level 2
        male1.addChild(male2);
        male2.setPartner(female3);

        male1.addChild(male3);
        male3.setPartner(female4);

        male1.addChild(female2);
        female2.setPartner(male6);

        //Level 3
        male2.addChild(male4);
        male4.setPartner(female6);

        male2.addChild(male8);
        male8.setPartner(female8);

        male3.addChild(female5);
        male3.addChild(male5);

        //Level 4
        male4.addChild(female7);

        male4.addChild(male7);
        male7.setPartner(female10);

        male8.addChild(female9);

        return root;
    }

    /**
     * Node/tree class inspired and heavily modified from:
     * http://stackoverflow.com/a/3522481
     * http://opendatastructures.org/
```

```
 * @author Nick Venenga et al. (links above)
 * @param <String> person's name
 */
@SuppressWarnings("hiding")
public static class Node<String> {
      private String data;
 private Node<String> parent;
 private Node<String> partner;
 private ArrayList<Node<String>> children;

 /**
  * Construct a new node with a name
  * and empty list of children
  * @param name of the person (the node represents)
  */
 public Node(String name) {
             data = name;
             children = new ArrayList<Node<String>>();
      }

 public String getName() {
      return data;
 }

 public void setPartner(Node<String> n) {
      partner = n;
 }

 /**
  * Adds a child to the node and
  * sets the child's parent as self
  * @param n child to add
  */
 public void addChild(Node<String> n) {
      children.add(n);
      n.parent = this;
 }

 /**
  * Gets the depth of a node in the tree
  * @param node to find depth of
  * @return depth of node
  */
 public int depth(Node<String> node) {
      Node<String> root = this;
      int d = 0;
      while (node != root) { //count parents
             node = node.parent;
             d++;
      }
      return d;
}

 /**
  * Finds a node in the tree
```

```
         * Modified from opendatastructures breadth-first search
         * @param name of node to find
         * @return node or null if not found
         */
        public Node<String> find(String name) {
             //make a pile of nodes
             Queue<Node<String>> q = new LinkedList<Node<String>>();

             //put the root in the pile
             if (this != null) q.add(this);

             //start looking through the pile
             while (!q.isEmpty()) {
                    //grab a node off the pile
                    Node<String> u = q.remove();

                    //have we found the node???
                    if(u.data.equals(name) || (u.partner != null &&
u.partner.data.equals(name))) return u;

                    //this isn't the node you're looking for
                    ///...so add some more
                    for(Node<String> c : u.children)
                          q.add(c);
             }

             //failure.
             return null;
       }

        /**
         * Finds a node then returns its depth
         * @param name of node to find
         * @return depth of node
         */
        public int findDepth(String name) {
             Node<String> n = find(name);
             if(n == null) return -1; //something went wrong (couldn't find node)
             return depth(n);
        }
    }
}
```

Prolog

Analysis

Sticking to the tool metaphor here, Prolog was like walking into a garage and finding a single wrench that worked perfectly. The program was relatively simple to write in Prolog (maybe 20-30 minutes after reviewing some old assignments and testing) and the recursive backtracker did all the work (whereas a lot of that work needed to be manually programmed into Scheme and Java). For logic problems Prolog is an excellent choice and can hugely ease development. In the future, where rule->outcome type programs are desired, I will definitely consider Prolog. In addition, Prolog has more documentation online and variants such as SWI-Prolog are being actively used and discussions can be found more readily than for Scheme. I did end up slightly modifying the example and just using "parent" instead of mother and father since that adds complexity to the code and the program still works correctly without the differentiation. Prolog does have a learning curve, but I think the practice in class and amount of online documentation allow anyone with a decent understanding of programming to learn the language.

Correctness

Clean and correct—no issues.

Output

```
yes:greatuncleaunt(male6,female7)
no:greatuncleaunt(male6,male8)
no:greatuncleaunt(female7,male3)
yes:greatuncleaunt(female4,female7)
no:greatuncleaunt(female4,male3)
yes:greatuncleaunt(male3,male7)
yes:greatuncleaunt(male6,female7)
no:greatuncleaunt(female7,male6)
no:greatuncleaunt(male5,female8)
no:greatuncleaunt(male5,male8)
no:greatuncleaunt(male8,female8)
```

Code

```
parent(male1, male2).
parent(female1, male2).
parent(male1, male3).
parent(female1, male3).
parent(male1, female2).
parent(female1, female2).

parent(male2, male4).
parent(female3, male4).
parent(male2, male8).
parent(female3, male8).

parent(male3, female5).
parent(female4, female5).
parent(male3, male5).
parent(female4, male5).

parent(male4, female7).
parent(female6, feamle7).
parent(male4, male7).
parent(female6, male7).

parent(male8, female9).
parent(female8, female9).

married(male1, female1).
married(male2, female3).
married(male3, female4).
married(female2, male6).
married(male4, female6).
married(male8, female8).
married(male7, female10).

married0(S, X) :- married(S, X).
married0(S, X) :- married(X, S).

siblings(X, Y) :- parent(Z, X), parent(Z, Y), \+ X = Y.
greatuncleaunt(X, Z) :- siblings(X, Y), parent(Y, A), parent(A, Z).
greatuncleaunt(S, Z) :- married0(S, X), siblings(X, Y), parent(Y, A), parent(A, Z).
```

Scheme

Analysis

If Java was a 500lb toolbox, Scheme was like going into the garage looking for a wrench and only finding a piece of wood and deciding to give that a go. Only to realize you're not holding a piece of wood, you're holding nothing. So then you wonder out into the yard and find a rock. As you're beating the bolt with the rock you realize you've actually somehow procured a limp spaghetti noodle instead. Metaphors aside, Scheme was painful. Development time was agonizingly long and the learning curve was high. I had to consult previous assignments and class information since online documentation was either non-existent or poor. In the end, my code has lots of similar code-blocks with slight modifications. In addition, there are patches in place to mitigate errors (such as adding a root to the family tree). In addition, the language isn't intuitive at all with its naming conventions. "Car" instead of "pop" can be a little confusing but is nothing compared to "caadddaaddadar". The standard library has the bare-minimum so it's necessary to create everything from scratch or copy-paste it out of previous assignments. My solution ended but being very poor although fully functional. Due to the way marriage relations are calculated, I have to call the getSiblings function again on itself to get proper marriage relations. In the process, the resulting list ends up having a huge number of duplicates. In summary, this language was a terrible chose and painful to create a program in. I don't think I would be able to efficiently create a solution in the future.

Correctness

It gets the job done but not well.

Output

```
(greatuncle "male6" "female7")
(greatuncle "male6" "male8")
(greatuncle "female7" "male3")
(greatuncle "female4" "female7")
(greatuncle "female4" "male3")
(greatuncle "male3" "male7")
(greatuncle "male6" "female7")
(greatuncle "female7" "male6")
(greatuncle "male5" "female8")
(greatuncle "male5" "male8")
(greatuncle "male8" "female8")
> #t
> #f
> #f
> #t
> #f
> #t
> #t
> #f
> #f
> #f
> #f
```

Code

;data structure below

```
(define parent
       '(
       ("root" "male1")
       ("root" "female1")
       ("male1" "male2")
       ("female1" "male2")
       ("male1" "male3")
       ("female1" "male3")
       ("male1" "female2")
       ("female1" "female2")

       ("male2" "male4")
       ("female3" "male4")
       ("male2" "male8")
       ("female3" "male8")

       ("male3" "female5")
       ("female4" "female5")
       ("male3" "male5")
       ("female4" "male5")

       ("male4" "female7")
       ("female6" "feamle7")
       ("male4" "male7")
       ("female6" "male7")

       ("male8" "female9")
       ("female8" "female9")
       )
)

(define marriage
       '(
       ("male1" "female1")
       ("male2" "female3")
       ("male3" "female4")
       ("female2" "male6")
       ("male4" "female6")
       ("male8" "female8")
       ("male7" "female10")
       )
)

; code begins on next page


; I shrunk the font size down—you'll thank me later
```

```scheme
(load "relations.scm")

;zmuda (some file)
(define (mydisplay value)
	(display value)
	(newline)
	#t
)
;zmuda list.scm
(define (ismember? atm lst)
	(cond
		((null? lst) #f)
		((equal? atm (car lst)) #t)
		(else (ismember? atm (cdr lst)))
	)
)

(define (join a b)
  (cond
	((null? b) a)
	(else (join (cons (car b) a) (cdr b)))
  )
)

(define (getParentsHelper person parent parents)
  (cond
     ((null? parent) parents)
     ((equal? person (cadar parent)) (append parents (list (caar parent)) (getParentsHelper person (cdr
parent) parents)))
     (else (getParentsHelper person (cdr parent) parents))
  )
)
(define (getParents person)
	(getParentsHelper person parent '())
)
(define (getParentsListHelper people parent parents)
	(cond
		((null? people) parents)
		(else(append parents (getParents (car people)) (getParentsListHelper (cdr people) parent
parents)))
	)
)
(define (getParentsList people)
	(getParentsListHelper people parent '())
)

(define (getChildrenHelper person parent children)
	(cond
		((null? parent) children)
		((equal? person (caar parent)) (append children (list (cadar parent)) (getChildrenHelper
person (cdr parent) children)))
		(else (getChildrenHelper person (cdr parent) children))
	)
)

(define (getChildren person)
	(getChildrenHelper person parent '())
)

(define (getSiblingsHelperBlood person parents siblings)
	(if (null? parents)
	(siblings)
	(append siblings
		(getChildren (car parents))
		(getChildren (cdr parents))
	)
	)
)
```

```
(define (getSiblingsHelperMarriage person marriage siblings)
        (cond
                ((null? marriage) siblings)
                ((equal? person (caar marriage)) (append siblings (list (cadar marriage))
(getSiblingsHelperMarriage person (cdr marriage) siblings)))
                ((equal? person (cadar marriage)) (append siblings (list (caar marriage))
(getSiblingsHelperMarriage person (cdr marriage) siblings)))
                (else (getSiblingsHelperMarriage person (cdr marriage) siblings))
        )
)

(define (getSiblings person)
        (join
                (if (null? (getParents person)) '() (getSiblingsHelperBlood person (getParents person)
'()))
                (getSiblingsHelperMarriage person marriage '())
        )
)
(define (getSiblingsListHelper people siblings)
        (cond
                ((null? people) siblings)
                (else(append siblings (getSiblings (car people)) (getSiblingsListHelper (cdr people)
siblings)))
        )
)
(define (getSiblingsList people)
        (getSiblingsListHelper people '())
)
; need getSiblingList twice to get married people correctly--the first pass won't get everyone (just blood
and married to the parameter)
(define (greatuncle person1 person2)
        (ismember? person1 (getSiblingsList (getSiblingsList (getParentsList (getParents person2)))))
)

(greatuncle "male6" "female7")
(greatuncle "male6" "male8")
(greatuncle "female7" "male3")
(greatuncle "female4" "female7")
(greatuncle "female4" "male3")
(greatuncle "male3" "male7")
(greatuncle "male6" "female7")
(greatuncle "female7" "male6")
(greatuncle "male5" "female8")
(greatuncle "male5" "male8")
(greatuncle "male8" "female8")
(mydisplay "test cases done")
,exit
```

Z+-

Java

Analysis

As is usually the case with Java, the language offers a lot of tools that provide many similar functions and it can be tricky and daunting trying to get what you want to work correctly. I originally intended to try to implement the interpreter using reflection but quickly realized I didn't have enough experience with reflection and the learning curve would take prohibitively long to overcome. Next, I started with a highly object oriented approach and tried to implement dynamic typing but once again realized I didn't have the correct experience to efficiently create a solution. Finally, I stuck with a method similar to how I created the C++ program but instead used regular expressions to match lines. Java's powerful regular expression engine made validating and parsing lines a breeze. I think it would have been interesting to try a context-free grammar approach, but most languages don't have good support. I think regular expressions are a good alternative although some may disagree. [1] [2] Overall, a decent amount of time was spent trying to formulate a good strategy given the wide array of options, but Java's superior built in string support made creating a parser significantly easier than C++. In addition, Java has a wealth of documentation online and it's very easy to find info on the language. The API documentation provided by Oracle is easy to read which is also helpful. Performance wise, Java has additional memory overhead and the overhead of running the virtual machine which could require more memory when interpreting sufficiently large programs. For the examples we used, there were no discernable differences. In addition, Java's exception made error handling much easier.

Correctness

Works per term project spec. Doesn't work with any loops.

Output

Loops removed from tests

```
e:\Files\Eclipse\Java\ZPM\bin>for /l %x in (10,1,15) do java zpm
..\samples\prog%x.zpm

e:\Files\Eclipse\Java\ZPM\bin>java zpm ..\samples\prog10.zpm
1
3
0

e:\Files\Eclipse\Java\ZPM\bin>java zpm ..\samples\prog11.zpm
33

e:\Files\Eclipse\Java\ZPM\bin>java zpm ..\samples\prog12.zpm
34Dogsaregood.

e:\Files\Eclipse\Java\ZPM\bin>java zpm ..\samples\prog13.zpm
Unknown operator *= for string on line 2

e:\Files\Eclipse\Java\ZPM\bin>java zpm ..\samples\prog14.zpm
Unknown operator -= for string on line 2

e:\Files\Eclipse\Java\ZPM\bin>java zpm ..\samples\prog15.zpm
44
```

Code

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class zpm {
	public static HashMap<String,Object> variableTable;
	public static final String varNameRules = "[a-zA-Z][a-zA-Z0-9-_]*";
	public static final String operatorRules = "[\\Q+-*\\E]?\\=";
	public static final Pattern varAssignment = Pattern.compile("^(" + 
varNameRules + ") (" + operatorRules + ") (" + varNameRules + ") ;$");
	public static final Pattern printStmt = Pattern.compile("^(print) (" + 
varNameRules + ") ;$", Pattern.CASE_INSENSITIVE);
	public static final Pattern strAssignment = Pattern.compile("^(" + 
varNameRules + ") (" + operatorRules + ") \"(.*?)\" ;$");
	public static final Pattern intAssignment = Pattern.compile("^(" + 
varNameRules + ") (" + operatorRules + ") (-?[0-9]+) ;$");

	//variable validity checker
	public static boolean variableExists(String name) {
		return variableTable.get(name) != null;
	}

	//variable creator
	public static void addVariable(String name, Object data) {
		variableTable.put(name, data);
	}

	//variable remover
	public static void deleteVariable(String name) {
		variableTable.remove(name);
	}

	//variable getter
	public static Object getVariable(String name) {
		Object o = variableTable.get(name);
		if(o == null)
			throw new IllegalArgumentException("Variable '" + name + "' does 
not exist");
		return o;
	}

	/**
	 * Converts a datatype into a string so it can replace variable references 
with their data
	 * @param name
	 * @return
	 * @throws Exception
	 */
	public static String getVariableString(String name) throws Exception {
		Object o = getVariable(name);
		if(o.getClass() == String.class) {
			return "\"" + (String)o + "\"";
		}
		else if(o.getClass() == Integer.class) {
			return ((Integer)o).toString();
		}
		else {
			throw new Exception("Unknown type " + o.getClass() + " for 
variable: " + name);
		}
	}
	
	public static void printVariable(String name) {
		System.out.println(name + " " + getVariable(name));
	}
	
	/**
	 * Handles all operations for integer data type
	 * @param variableName
	 * @param operation
	 * @param data
	 */
	public static void doIntegerOperation(String variableName, String operation, 
String data) {
		try {
			Integer temp = Integer.parseInt(data);
			if(operation.equals("=")) {
				//do nothing data already contains correct value
			}
			else if (operation.equals("-=")) {
				temp -= (Integer)getVariable(variableName);
			}
			else if (operation.equals("+=")) {
				temp += (Integer)getVariable(variableName);
			}
			else if (operation.equals("*=")) {
				temp *= (Integer)getVariable(variableName);
			}
			else {
				throw new IllegalArgumentException("Unknown operator " + 
operation + " for integer");
			}
			if(variableExists(variableName))
				deleteVariable(variableName);
			addVariable(variableName, (Object) temp);
		} catch (Exception e) {
			throw new IllegalArgumentException("Error " + data + " is not of 
type integer");
		}
	}
	
	/**
	 * Handles all operations for string data type
	 * @param variableName
```

```
         * @param operation
         * @param data
         */
        public static void doStringAssignment(String variableName, String operation,
String data) {
                String temp = data;
                if(operation.equals("=")) {
                        //straight assignment, do nothing to temp
                }
                else if(operation.equals("+=")) {
                        try {
                                temp = (String)getVariable(variableName);
                        }
                        catch (Exception e) {
                                throw new IllegalArgumentException(variableName + " is not
of type string");
                        }
                        temp += data;
                }
                else {
                        throw new IllegalArgumentException("Unknown operator " +
operation + " for string");
                }

                if(variableExists(variableName))
                        deleteVariable(variableName);
                addVariable(variableName, (Object) temp);
        }

        public static void parseLine(String line) throws Exception {
                Matcher m;

                //replace variable with its corresponding data
                if(varAssignment.matcher(line).matches()) {
                        m = varAssignment.matcher(line);
                        m.find();
                        String variableName = m.group(1);
                        String operation = m.group(2);
                        String variableName2 = m.group(3);
                        //replace variable name with data
                        parseLine(variableName + " " + operation + " " +
getVariableString(variableName2) + " ;");
                }
                //check if the line looks like a string
                else if (strAssignment.matcher(line).matches()) {
                        m = strAssignment.matcher(line);
                        m.find();
                        doStringAssignment(m.group(1), m.group(2), m.group(3));
                }
                //check if line looks like an integer
                else if (intAssignment.matcher(line).matches()) {
                        m = intAssignment.matcher(line);
                        m.find();
                        doIntegerOperation(m.group(1), m.group(2), m.group(3));
                }
```

```
            //check if line contains a print statement
            else if (printStmt.matcher(line).matches()) {
                  m = printStmt.matcher(line);
                  m.find();
                  System.out.println(getVariable(m.group(2)));
            }
            else {
                  throw new RuntimeException("Parser doesn't understand " + line);
            }
      }

      public static void main(String[] args) throws IOException {
            //fail if not input file is provided
            if(args.length != 1) {
                  System.err.println("Usage: java zpm program.zpm");
            }

            //initialize the variable table
            variableTable = new HashMap<String, Object>();

            //open the input file
            BufferedReader br = null;
            try {
                  br = new BufferedReader(new FileReader(args[0]));
            } catch(Exception e) {
                  System.err.println("Error opening: " + args[0]);
                  System.exit(1);
            }

            String line = null;
            int lineNumber = 1;

            //loop through every line of the input file
            while((line = br.readLine()) != null) {
                  try {
                        //run the input line through the parser
                        parseLine(line);
                  }
                  catch (Exception e) { //cleanup any exception with error text
instead
                        System.err.println(e.getMessage() + " on line " +
lineNumber);
                        System.exit(1);
                  }
                  lineNumber++;
            }
      }

}
```

C++

Analysis

C++ isn't terribly suited for the task, but it's very unforgiving. The standard library has little support for advanced string processing and it's expected to use a comprehensive third-party solution like Boost. Whereas Java gives you every tool imaginable to solve a problem, C++ gives you a foundation where you can build any tool you need. This can be either a positive or negative depending on the application. In this case, C++ took much more time to create a solution in since many string processing techniques had to be created instead of already existing. Luckily there is a lot of C++ documentation online, but some can be quite old dating back 10+ years and reference older versions of C++ or straight C. Performance wise, C++ is likely much faster than Java (in regards to memory usage in this application). On the other hand, development time is significantly greater so it's a balance between the performance the code needs and the time a developer is willing to spend writing the program. I think the language definitely isn't ideal. It might be better with 3rd party libraries that provide better string tools, but by default, it's a challenge. Error handling in C++ made it more difficult to pass errors back up. I ended up using an enum to keep track of error codes but pretty much every function returned a series of ints until they finally reached the top where they were reported.

Correctness

The program works per spec and with non-nested loops. Nested loops output incorrect results.

Output

```
e:\Files\Eclipse\C++\Z+-\Debug>for /l %x in (10,1,15) do z+- ..\samples\prog%x.zpm

e:\Files\Eclipse\C++\Z+-\Debug>z+- ..\samples\prog10.zpm
A=1
a=3
numItems=0

e:\Files\Eclipse\C++\Z+-\Debug>z+- ..\samples\prog11.zpm
A=33

e:\Files\Eclipse\C++\Z+-\Debug>z+- ..\samples\prog12.zpm
A=34Dogsaregood.

e:\Files\Eclipse\C++\Z+-\Debug>z+- ..\samples\prog13.zpm
ERROR: Parse Error on line 2
 ->A *= A ;

e:\Files\Eclipse\C++\Z+-\Debug>z+- ..\samples\prog14.zpm
ERROR: Parse Error on line 2
 ->A -= "1" ;

e:\Files\Eclipse\C++\Z+-\Debug>z+- ..\samples\prog15.zpm
A=4444444444444444
```

Code

```
/*
 * utils.h
 *
 *  Created on: Jan 28, 2016
 *      Author: Nick
 */

#include <algorithm>
#include <functional>
#include <cctype>
#include <locale>

using namespace std;

#ifndef UTILS_H_
#define UTILS_H_

static inline string& ltrim(string&);
static inline string& rtrim(string&);
static inline string& trim(string&);

#endif /* UTILS_H_ */

// from http://stackoverflow.com/a/217605/2751619
// trim from start
static inline string &ltrim(string &s) {
        s.erase(s.begin(), find_if(s.begin(), s.end(), not1(ptr_fun<int,
int>(isspace))));
        return s;
}

// trim from end
static inline string &rtrim(string &s) {
        s.erase(find_if(s.rbegin(), s.rend(), not1(ptr_fun<int,
int>(isspace))).base(), s.end());
        return s;
}

// trim from both ends
static inline string &trim(string &s) {
        return ltrim(rtrim(s));
}
//============================================================================
// Name        : Z+-.cpp
// Author      : Nick Venenga
// Version     :
// Copyright   :
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
#include <fstream>
#include <unordered_map>
```

```
#include <string.h>
#include <vector>
#include <exception>
#include "utils.h"
#include <stdio.h>
#include <ctype.h>

using namespace std;

int parseLine(string);
int line; // keep track of line of program interpreter is on
bool debug; // keep track of whether debug mode is on

// Keep track of variable types
enum zTypes {
	zInt,
	zStr
};

// Keep track of statuses for each interpreted command
enum zStatusCodes {
	Z_OK,
	Z_PARSEERROR,
	Z_UNMATCHEDQUOTES,
	Z_LOGICERROR,
	Z_TYPEERROR,
	Z_CRASH,
	Z_NAMEERROR,
};
string zStatusText[] = {"OK", "Parse Error", "Unmatched Quotes", "Logic Error", "Type
Error", "Unexpected failure occurred", "Illegal name-variables must start with alpha
character"};


// Class to keep track of variables
//**in a perfect world this would have used
//  dynamic types in C++ and interpreted Z+-
//  into C
class variable {
	int type;
	int intVal;
	string strVal;

	public:
	variable(string val) {type = zStr; intVal = 0; strVal = val;}
	variable(int i) {type = zInt; intVal = i; strVal = "";}
	int getType() {return type;}
	void setType(int t) {type=t;}
	string getStrValue() {return strVal;}
	void setStrValue(string s) {strVal=s;}
	int getIntValue() {return intVal;}
	void setIntValue(int i) {intVal = i;}
	string str() {
		return (type == zInt) ? to_string(intVal) : strVal;
	}
	string info() { // for debugging
		return ((type == zInt) ? "(int)" : "(str)") + str();
	}
};

//Create a map to keep track of variables
unordered_map<string,variable*> variableTable;

//Adds a variable to the variable table
void addVariable(string name, variable* v) {
	variableTable.insert({name, v});
}

//Dumps variable table for debugging
void debugVariableTable() {
	for(auto item : variableTable) {
	   cout << item.first << ": " << item.second->info() << endl;
	}
}

//Checks if a variable exists
bool varExists(string varName) {
	bool result = variableTable.count(varName) == 1;
	if(debug)
		cout << "Checking if " << varName << " exists " << result << endl;
	return result;
}

//Get a variable pointer given a name
//**invalid name crashes interpreter with exception
variable* getVariable(string name) {
	unordered_map<string,variable*>::const_iterator got =
variableTable.find(name);
	if(got == variableTable.end()) {
		string error = "Variable not found @";
		error += line;
		error += ": ";
		error += name;
		throw domain_error(error);
	}
	return got->second;
}

//Delete a variable
void deleteVariable(string name) {
	variable* v = getVariable(name);
	variableTable.erase(name);
	delete v;
}

//Variable assignments are handled here
int doAssignment(string name, string value, int type) {
	variable* newVar;

	name = trim(name);
	//Delete variable if it already exists
	if(varExists(name)) {
		deleteVariable(name);
	}

	if(type == zStr) {
		newVar = new variable(value);
	}
	else if (type == zInt) {
		newVar = new variable(stoi(trim(value)));
	}

	if(debug)
		cout << "Adding " << name << " value:" << value << endl;

	addVariable(name, newVar);
	return Z_OK;
}

//Variable mutators are handled here
int doOperation(variable* tempVar, char op, string value) {
//Decide what to do based on operand
	int intVal;
	string strVal;

	value = trim(value);

	try { // do something dangerous :O
		switch(tempVar->getType()) {
		case zInt:
			if(varExists(value))
				intVal = getVariable(value)->getIntValue();
			else
				intVal = stoi(value);
			break;
		case zStr:
			if(varExists(value))
				strVal = getVariable(value)->getStrValue();
			else
				strVal = value;
			break;
		default:
			return Z_CRASH;
		}
	}
	catch (invalid_argument &e) {
		return Z_TYPEERROR; //happens when stoi fails on non-int
	}


	switch(op) {
	case '+': // += assignment
		switch(tempVar->getType()) {
		case zInt:
```

```
                    tempVar->setIntValue(tempVar->getIntValue() + intVal);
                    break;
            case zStr:
                    tempVar->setStrValue(tempVar->getStrValue() + strVal);
                    break;
            default: // Operation undefined for datatype
                    return Z_PARSEERROR;
            }
            break;
        case '*': // *= assignment
            switch(tempVar->getType()) {
            case zInt:
                    tempVar->setIntValue(tempVar->getIntValue() * intVal);
                    break;
            default: // Operation undefined for datatype
                    return Z_PARSEERROR;
            }
            break;
        case '-':// -= assignment
            switch(tempVar->getType()) {
            case zInt:
                    tempVar->setIntValue(tempVar->getIntValue() - intVal);
                    break;
            default: // Operation undefined for datatype
                    return Z_PARSEERROR;
            }
            break;
        }

        return Z_OK;
}

// Parse a line multiple times (loop)
int doLoop(string line, int times) {
        int status; // keep track of execution status
        for(int i = 0; i < times; i++) {
            status = parseLine(line);
            if(status != Z_OK) // Stop on error
                    return status;
        }
        return Z_OK;
}

int parseLine(string line) {
        if(debug)
            cout << "PARSING: " << line << endl;

        line = trim(line);
        string upper = line;
        transform(upper.begin(), upper.end(), upper.begin(), ::toupper);
        size_t eqPos = line.find("=");
        variable* tempVar;
        string name, command, value;

        // Handle FOR/loop command
        if(line.substr(0, 3) == "FOR") {
            //Check for ENDFOR
            if(line.substr(line.length()-6, line.length()) != "ENDFOR")
                    return Z_PARSEERROR;
            //Get iteration count
            int space = line.find(' ', 4);
            int count = stoi(line.substr(4, space));
            //Process loop (removing FOR and ENDFOR)
            space = line.find(' ', space);
            command = line.substr(space+1, line.length()-6-space-1);
            return doLoop(command, count);
        }
        else {
            //Figure out how many statements are on the line
            bool inQuotes = false;
            int stmtCount = 0;
            int lastSemiColon = 0;
            vector<string> stmts;
            for(int i = 0; i < (int)line.length(); i++) {
                    switch(line.at(i)) {
                    case 'F':
                            if(!inQuotes && upper.substr(i, 3) == "FOR")
                                    inQuotes = !inQuotes;
                            break;
                    case 'R': // Check for 'ENDFOR' for nested loops and consider a
';'
                            if(!inQuotes && strcmp(line.substr(line.length()-6,
6).c_str(), "ENDFOR") != 0) {
                                    inQuotes = !inQuotes;
                                    i += 7;
                            }
//                              break;
                    case ';': // Found another statement
                            if(!inQuotes) { // Make sure it's not part of a string
                                    stmtCount++; // Found another statement
                                    command = line.substr(lastSemiColon, i-
lastSemiColon+1);

                                    stmts.push_back(command); // Add statement to
statement list

                                    lastSemiColon = i+2; // Change location of beginning
of new statement
                            }
                            break;
                    case '"': // Found a string
                            inQuotes = !inQuotes;
                            break;
                    }
            }
            if(debug)
                    for(auto cmd : stmts) {
                            cout << "PARSING MULTI-STMT: " << cmd << endl;
                    }

            // Handle multiple statements
            if(stmtCount > 1) {
```

```
		for(int i = 0; i < stmtCount; i++) {
			int status = parseLine(stmts.at(i));
			if(status != Z_OK)
				return status;
		}
		return Z_OK; // weren't any errors above so O.K.
	}

	// Lines must end in ;
	if(line.back() != ';')
		return Z_PARSEERROR;

	// Found an equal sign, handle some sort of assignment
	if (eqPos != string::npos) {
		int variableType;

		// Get the variable name
		name = string(line.substr(0, line.find(' ')));
		name = rtrim(name);
		string right_side = line.substr(eqPos+1, line.length()-eqPos);
		string right_var = right_side.substr(1, right_side.length()-3);

		//Check variable naming
		if(!isalpha(name.c_str()[0]))
			return Z_NAMEERROR;

		// Figure out the data type
		if(varExists(right_var)) {
			variableType = (zTypes)(getVariable(trim(right_var))-
>getType());
			value = getVariable(trim(right_var))->str();
		}
		else if(right_side.at(1) == '"') {
			variableType = zStr;
			value = right_side.substr(2, right_side.length()-5);
		}
		else {
			variableType = zInt;
			value = right_var;
		}

		// Do assignment or mutator operation
		if(line.at(eqPos-1) == ' ') { // straight assignment
			return doAssignment(name, value, variableType);
		}
		else { //composite assignment
			tempVar = getVariable(name);
			return doOperation(tempVar, line.at(eqPos-1), value);
		}
	}
	// No equal sign, check for a command/reserved word
	else if (upper.substr(0, 5) == "PRINT"){
		name = line.substr(6, line.length()-6-2);
		cout << name << "=" << getVariable(name)->str() << endl;
	}
```

```
		else {
			return Z_PARSEERROR;
		}
	}
	return Z_OK; //temporary
}

void failMessage(string name) {
	cout << "Usage " << name << " [-d] program.zpm" << endl;
}

int main(int argc, char *argv[]) {
	int status; // keep track of status of each line (ok, error, etc)
	debug = false;
	line = 1; // initialize current script line

	if(argc >= 2) {
		// Check command line args
		// -> no more than 3 args
		// -> if 3 args, arg 1 must be -d (debug switch)
		if((argc == 3 && (strcmp(argv[1], "-d") != 0)) || argc > 3) {
			failMessage(argv[0]);
			return Z_PARSEERROR; // couldn't parse file
		}
		if(argc == 3)
				debug = true; //turn debug mode on
	}
	else {
		failMessage(argv[0]);
		return Z_CRASH;
	}

	//http://www.cplusplus.com/forum/beginner/24492/
	ifstream ifs(argv[argc-1]);
	string str;
	while(getline(ifs, str)) {
		if(debug)
			cout << str << endl;
		status = parseLine(str);
		if(status != Z_OK) {
			cout << "ERROR: " << zStatusText[status] << " on line " << line
<< endl;
			cout << " ->" << str << endl;
			return status; // return relevant error code for debugging
		}

		line++;
	}

	if(debug)
		debugVariableTable();
	return 0;
}
```

Nearest Neighbor

C++

Analysis

Those pointers and vectors and vectors of pointers and pointers to vectors of pointers pointing to pointers. Once I got that figured out, writing the program was pretty simple. C++ excels at doing computations quickly and the resulting program is relatively simple and significantly faster than Python. The majority of development time was spent understanding how vectors and their iterators worked—especially since I had a vector of vectors and that was really throwing me off. There is a learning curve to C++, but I've also been spending a decent amount of time creating code for other problems and assignments so once I got the pointer/vector situation figured out it was pretty simple. C++ is definitely the best choice for this task if you need performance. Running the prototypes against themselves (about 4000*4000 combinations) C++ finished in 1m30s and used around 10mb of memory. Python finished in 15m30s and used 20mb of memory. C++ is the clear winner here unless a short one-off program is needed. Even though Python is quite portable, C++ standard library was used so the program should compile on most platforms without issue.

Correctness

Tested prototypes against themselves and got 100% accuracy so seems like it's good.

Output

```
DATA FILE =  /home/venengnj/zmudaTP/data/nn/bupa
1
1
2
2
2
1
2
1
1
1
DATA FILE =  /home/venengnj/zmudaTP/data/nn/spam
2
2
```

Code

```
//============================================================================
// Name        : nn.cpp
// Author      : Nick Venenga
// Description : Comparative programming linearly nearest neighbor
//============================================================================

#include <float.h>
#include <stdlib.h>
#include <cmath>
#include <fstream>
#include <iostream>
#include <sstream>
#include <iterator>
#include <string>
#include <vector>

using namespace std;

vector<vector<double>* > fileToVector(string filename) {
	string line;
	vector<vector<double>* > values = vector<vector<double>* >();

	ifstream file (filename.c_str());
	if(file.is_open()) {
		while(getline(file, line)) {
			istringstream strstream(line);
			string token;
			vector<double>* line_vals = new vector<double>();
			while(getline(strstream, token, ',')) {
				line_vals->push_back(atof(token.c_str()));
			}

			values.push_back(line_vals);
		}
		file.close();
	}

	return values;
}

int nearestNeighbor(vector<vector<double>* >* prototypes, vector<double>* item) {
	double minDistance = DBL_MAX;
	int result = -1;

	double sum;
	double distance;
	vector<vector<double>* >::iterator row;

	for(row = (*prototypes).begin(); row != (*prototypes).end(); ++row) { //loop
through each prototype item
		sum = 0; //reset sum
		distance = DBL_MAX; //reset distance
		for(unsigned int i = 0; i < (*row)->size()-1; i++)
			sum += pow(item->at(i) - (*row)->at(i), 2); // add the sum of
each pair squared (for distance)

		distance = pow(sum, .5); // get the sq root

		if(distance < minDistance) {
			minDistance = distance; // new minimum distance
			result = (*row)->at((*row)->size()-1); // result is the last item
in the row
		}
	}
	return result;
}

void vectorCleanup(vector<vector<double>* > v) {
	vector<double>* ptr;
	while(!v.empty()) {
		ptr = v.back();
		v.pop_back();
		delete ptr;
	}
}

int main(int argc, char* argv[]) {
	string prototypesFile = "bupaPrototypes.txt";
	string unknownsFile = "bupaUnknowns.txt";

	if(argc == 3) {
		prototypesFile = argv[1];
		unknownsFile = argv[2];
	}

	vector<vector<double>* > prototypes = fileToVector(prototypesFile);
	vector<vector<double>* > unknowns = fileToVector(unknownsFile);

	vector<vector<double>* >::iterator row;
	for(row = unknowns.begin(); row != unknowns.end(); ++row) {
		cout << nearestNeighbor(&prototypes, *row) << endl;
	}

	//cleanup memory instead of leaving it around (makes valgrind happy)
	vectorCleanup(prototypes);
	vectorCleanup(unknowns);

	return 0;
}
```

Python

Analysis

Like usual, Python was a breeze to write the program in and it took only about 10-15 minutes. Python's comprehensive string built-in string library makes reading and performing simple operations on text files child's play. In addition, Python is currently a hugely popular language and there's a wealth of recent and up-to-date information online. Not only that, Python is designed to minimize the possible ways to complete a task to increase the simplicity of the language. Where Java is like the 500lb tool chest, Python is like a small toolbox that usually has what you're looking for inside. In this case though, were large computations can come into play, I think Python was less than optimal unless development time is a bigger concern than performance.

Correctness

Tested prototypes against themselves with 100% accuracy. Seems good.

Output

```
DATA FILE =  /home/venengnj/zmudaTP/data/nn/bupa
1
1
2
2
2
1
2
1
1
1
DATA FILE =  /home/venengnj/zmudaTP/data/nn/spam
2
2
```

Code

```
#!/usr/bin/python3

import sys

prototypesFile = "bupaPrototypes.txt"
unknownsFile = "bupaUnknowns.txt"

if len(sys.argv) == 3:
	prototypesFile = sys.argv[1]
	unknownsFile = sys.argv[2]

prototypes = []
unknowns = []

with open(prototypesFile, 'r') as f:
	lines = f.readlines()
	prototypes = [line.rstrip().split(',') for line in lines]

with open(unknownsFile, 'r') as f:
	lines = f.readlines()
	unknowns = [line.rstrip().split(',') for line in lines]

def nearestNeighbor(prototypes, item):
	minDistance = float("inf") # set it large to start out (infinity in this case)
	result = -1

	for p in prototypes:
		sum = 0
		for i in range(len(p)-1):
			sum += (float(p[i]) - float(item[i]))**2
		distance = sum**(.5)
		if distance < minDistance:
			minDistance = distance
			result = p[-1]

	return result

for u in unknowns:
	print(str(nearestNeighbor(prototypes, u)))
```