Stuart Wyse
CSE 465 – Term Project
Language Analysis

Table of Contents

Zipcodes

C#

This language allowed me to solve the zipcode problem(s) fairly well. Given that I don't know C# very well, the main difficulty with this problem was just learning C#. Once I did that, I realized that C# had the features I needed to properly solve this problem. However, now that I have some experience with C#, I am confident that I would be able to solve problems more quickly with it in the future. I will absolutely still need a little refresher on C#, but I would hopefully be able to do that much more quickly. The run-time performance of C# was not noticeable different than Python in this problem. Overall, I would say the language is a reasonable choice, for me, with this problem. It's reasonable given that I was able to solve the problem with C#, but did it more quickly with Python. I was able to solve this more quickly with Python mainly because I know and understand Python better.

Python

With Python, I was able to solve this problem without much of a hassle. However, there seems to be an infinite loop in my CityStates problem that I was unable to solve. I do not understand why or how an infinite loop is possible in it, but there is one in there. Even with that, the output file is still populated correctly. Python provided better features for this problem, since it allows for much easier reading & writing to files. The ability to read in a file and strip the data line by line was very helpful. I also thought that being able to split a line was helpful when searching for certain elements of the data. For the reasons just listed, I would say I noticed an appreciable difference with Python over C# when solving this problem. I already had a fairly solid background in Python, so there wasn't as much of a learning curve as there was with C#. That being said, I think that I would be able to solve problems slightly quicker with Python in the future after solving this problem. As mentioned in the C# analysis, there was no noticeable run-time difference between the two languages. I would say that Python is the ideal language for this

problem. The great write ability Python has when it comes to reading in and splitting up data from a file makes it the ideal language for this problem. It is especially easier than C# when it comes to reading in a file, as I used a StreamReader in C#.

Summation

Java

I was able to solve the summation problem using Java. Java had the features I needed to solve the problem. Essentially, I just needed something to store the list of values in. There was a significant difference in development time with Java over Scheme or Prolog when solving this problem. A huge part of that is simply because Java is the language I know and understand the best. For this reason, there wasn't much of a learning curve with Java. When it comes to the run-time speed, I would say Java's was slightly slower than Scheme and Prolog. By no means was it *inefficient* though. Although the run-time was slightly slower, I would say Java is a reasonable language (of the three used to solve this problem) for me. With Scheme and Prolog, there was a large learning curve that I had to get past before I could even begin. As I went on, I had to continually learn how to implement the different aspects necessary to solve the problem. With Java, I just had to figure out the best way to solve the problem. I didn't have to research much about how to actually implement things. For example, in Java I could easily write code that would sum up a list. In Scheme, this was more difficult for me to figure out and I had to do a little bit of research from classes earlier this semester. Even with all of that, I chose Prolog as the ideal language because once you get past the slight learning curve, it is a very simple language and has a quicker run-time.

Prolog

I was able to fully solve the summation problem using Prolog. It took me some time, but once I was able to implement the sublst() function, I was able to get it working. Prolog had everything I needed to implement a correct solution. Being able to split up a list with the | (pipe) was essential to solving this problem. It really didn't take me long to implement the entire solution. The main thing that took time was understanding Prolog – I really had to test a lot of different things to understand it better. Now that I do understand Prolog better, I am confident that I could solve similar problems more quickly in the future. I think I would still have to try a lot of things out before completing a problem, however. By that I mean create small Prolog functions and test them with different inputs, in order to understand them better. The run-time for Prolog is fairly quick. Overall, I would say Prolog is the ideal language for this problem. As I mentioned in the Java section, Prolog had more of a learning curve, but once I got past that it was simple to implement a solution. Prolog also has a quicker run-time than Java.

Scheme

I came close to fully solving the summation problem using Scheme, however it was not 100% correct. It currently returns true if the target is in the list, or if the target can be reached by summing up n numbers from left to right. By that I mean that (1, 2, 3, 4, 5) with a target of 10 would be true. But (1, 2, 3, 5, 4) with a target of 10 would NOT be true, even though it should be true. The reason it doesn't return the correct answer is because the numbers are only summed up from left to right, and 1+2+3+5 = 11. It definitely took me a little longer to solve this problem in Scheme than it did with Java, given that there was more of a learning curve with Scheme. If I were to do similar problems in the future, I'm sure I would be able to do so a little quicker. I would absolutely still need to learn and research more, because Scheme is a language I have a tougher time understanding. Scheme's run-time to solve these summation problems was rather quick, and slightly faster than Java. Since its run-time is so fast, Scheme is a good language for this problem. However, since I am less competent with Scheme, I would say that it is reasonable. If I knew Scheme better, it would most likely be the ideal language. While testing, I also realized that when the

target is 0 and false should be returned, it's possible that the result will actually come back as 'unspecified'. I am unsure as to why that is happening. I think part of it has to do with the fact that I did not use cond(). I attempted to use cond(), knowing that it would work more efficiently, but I ended up creating more issues and was unable to implement a solution using cond().

Z+-

Java

I was able to solve the Z+- problem using Java. I made some fairly significant changes from the original Z+- homework to completely solve the problem. I also did not have any FOR loops working in the original homework, but I was able to implement that for the project. Java provided everything I needed to solve this problem. I used Java's HashMap to store variables and their values and to update them properly, and I just used a Scanner to read in the Z+- file. I would definitely be able to solve this problem or similar problems more quickly in the future. Even though I know Java well, I wasn't sure of what data structure I should use when I first started. Originally, I did not use a HashMap. So it took some time to implement that part of the solution. I did not notice any issue with the run-time of Java to solve this problem. Of Java and C++, Java is by far the ideal language for me to use to solve this problem. I understand and I am able to write Java much more quickly and efficiently than C++.

C++

I have most of this problem solved using C++. Really the only part I was unable to figure out was FOR loops. This took me some time to figure out in Java, and I unfortunately could not get it working in C++. There is also a Segmentation Fault occurring when I run the test script. C++ has everything I need to solve this problem. The development time was definitely slower than the time it took me to develop a solution in Java. Although my development time was slower in C++, I would be able to develop slightly more quickly in the future with similar problems. This is because I was able to work through a number of issues while solving this problem, and I would hopefully be able to avoid those issues next time. The run-time for this problem in C++ was good. I didn't notice a significant difference in the run-times of the Java and C++ programs. Overall, I would say C++ is a reasonable language to solve this problem. As I mentioned in the Java section, I am able to write Java code more easily than C++, making Java my ideal language for this problem. That being said though, C++ can still be used to come up with a solution.

Band Matrix

C++

I was able to solve most of the Band Matrix using C++, but it is not fully complete. I get some errors when running the test script with my solution as well. C++ provided all of the necessary features I needed to implement a solution. I just do not know C++ well enough to actually use all of those features, unfortunately. It took me a while to develop the solution I have. I do not know if I could solve this problem much quicker in the future, as I found it very challenging for me. It is difficult for me to talk much about the run-time, since there are errors when I run the program. C++ is definitely a poor choice for me with this problem. Overall, I had a very difficult time working with C++ on this problem. A large part of that is because I am used to Java, where I do not need to worry about memory management as much. So, when it comes to C/C++, I have a difficult time working with pointers, addresses, and memory management.

Python

Unfortunately, I was unable to implement anything for Python and the band matrix problem. The other problems took me longer to implement, and I was unable to attempt this one. That being said, I will do my best to analyze Python in regards to this problem. Python does have the necessary features to implement this problem. As for the development time, I would guess that I would be able to implement a solution more quickly in Python than I did in C++. The run-time performance of Python would probably be slightly slower than C++. This would be a reasonable language to use for this problem. It is definitely possible to implement a solution, but it is difficult for me to say it is ideal without implementing a solution.

Appendix

zipcodes.cs
SoC: All requirements were completed for this problem in this language.

```
// Stuart Wyse
// CSE 465 - Term Project
// zipcodes.cs

using System.IO;
using System;
using System.Collections.Generic;

class Program
{
 static void Main() {
  CommonCityNames();
  LatLon();
  CityStates();
 }

 //---------------------------------------------------------
 // CommonCityNames.txt

 static void CommonCityNames() {
  string line;
  System.IO.StreamReader file = new System.IO.StreamReader("states.txt");

  // List to hold states in states.txt
  List<string> states = new List<string>();

  // add states to states list
  while((line = file.ReadLine()) != null) {
   states.Add(line);
  }

  System.IO.StreamReader zipcodes = new System.IO.StreamReader("zipcodes.txt");
  List<string> CommonCityNames = new List<string>();

  // if city is in state that is in states.txt, add it to CommonCityNames
```

```
while((line = zipcodes.ReadLine()) != null) {
  string[] words = line.Split('\t');
  foreach(string state in states) {
    if(words[4] == state) {
      CommonCityNames.Add(words[3]);
    }
  }
}

// sort List, and write List to CommonCityNames.txt
CommonCityNames.Sort();
using(StreamWriter writeText = new StreamWriter("CommonCityNames.txt")) {
  CommonCityNames.ForEach(delegate(string city) {
    writeText.WriteLine(city);
    });
  }

  file.Close();
  zipcodes.Close();
}

// ------------------------------------------------
// LatLon.txt

static void LatLon() {
  string line2;
  System.IO.StreamReader file2 = new System.IO.StreamReader("zips.txt");

  // List to hold states in states.txt
  List<string> zips = new List<string>();

  // add states to states list
  while((line2 = file2.ReadLine()) != null) {
    zips.Add(line2);
  }

  System.IO.StreamReader zipcodes2 = new System.IO.StreamReader("zipcodes.txt");
  List<string> LatLon = new List<string>();

  // add zip, lat, and lon to LatLon list
  // use zip to make sure no duplicates are added
  while((line2 = zipcodes2.ReadLine()) != null) {
    string[] words2 = line2.Split('\t');
    foreach(string zip in zips) {
      if(words2[1] == zip && !LatLon.Contains(words2[1])) {
        LatLon.Add(words2[1]);
        LatLon.Add(words2[6]);
        LatLon.Add(words2[7]);
      }
    }
  }
```

```
  using (StreamWriter writeText2 = new StreamWriter("LatLon.txt")) {

    // add lat and lon to LatLon.txt
    int counter = 0;
    foreach(string entry in LatLon) {
      if(counter == 1 || counter == 2) {
        writeText2.Write(entry + ' ');
      }
      if(counter >= 2) {
        writeText2.Write('\n');
        counter = 0;
      } else {
        counter++;
      }
    }
  }
  file2.Close();
  zipcodes2.Close();
}

// ----------------------------------------------------
// CityStates.txt

static void CityStates() {
  string line3;
  System.IO.StreamReader file3 = new System.IO.StreamReader("cities.txt");

  // List to hold cities in cities.txt
  List<string> cities = new List<string>();

  // add cities to cities list
  while((line3 = file3.ReadLine()) != null) {
    cities.Add(line3);
  }

  System.IO.StreamReader zipcodes3 = new System.IO.StreamReader("zipcodes.txt");
  List<string> CityStates = new List<string>();

  // for each city, check each line of zipcodes for matching city
  // if city matches, add it to CityStates
  foreach(string city in cities) {
    // move to beginning of zipcodes.txt for each city
    zipcodes3.DiscardBufferedData();
    zipcodes3.BaseStream.Seek(0, System.IO.SeekOrigin.Begin);

    CityStates.Add("\n");
    CityStates.Add(city);
    while((line3 = zipcodes3.ReadLine()) != null) {
      string[] words3 = line3.Split('\t');
      // add states to CityStates list if they have the given city
```

```
      if(words3[3] == city.ToUpper()) {
        CityStates.Add(words3[4]);
      }
     }
    }

    List<string> CityStates2 = new List<string>();

    using (StreamWriter writeText3 = new StreamWriter("CityStates.txt")) {

     foreach(string entry in CityStates) {
      // if entry is the name of the city
      if(entry.Length != 2) {
        CityStates2.Sort();
        foreach(string cityState in CityStates2) {
          writeText3.Write(cityState + " ");
        }
        // clear the list for the next city
        CityStates2.Clear();
        writeText3.Write(entry);
        writeText3.Write(" ");

      } else if(entry.Length == 2 && !CityStates2.Contains(entry)) {
        CityStates2.Add(entry);
      }
     }
     // at end of CityStates, sort and write CityStates2
     CityStates2.Sort();
     foreach(string cityState in CityStates2) {
      writeText3.Write(cityState + " ");
     }

    }
    file3.Close();
    zipcodes3.Close();
  }
}
```

```
  GNU nano 2.2.6                File: CommonCityNames.txt                        Modified

ABERDEEN
ACME
ADA
ADA
ADA
ADA
ADA
ADAIRO
ADAMS COUNTY
ADAMS MILLS
ADAMSVILLE
ADDISON
ADDISON TOWNSHIP
ADDISON TWP
ADDYSTON
ADELPHI
ADENA
ADRIAN
ADRIAN
AFTON
AHMEEK
AK STEEL
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
```

```
  GNU nano 2.2.6                File: LatLon.txt

41.51 -84.3
39.5 -84.74
42.28 -83.17
```

```
 GNU nano 2.2.6                    File: CityStates.txt

Oxford AL AR CO CT FL GA IA ID IN KS MA MD ME MI MS NC NE NJ NY OH PA WI
Melvindale MI
Franklin AL AR CT GA IA ID IL IN KS KY LA MA MD ME MI MN MO NC NE NH NJ NY OH PA SD TN TX VA VT WI WV
```

---

zipcodes.py
SoC: There appears to be an infinite loop in the cityStates() method. Therefore, the output for that is not there. There are also duplicate latitudes and longitudes being put into the LatLon.txt file.

```
# Stuart Wyse
# CSE 465 - Term Project
# zipcodes.py

def main():
   commonCityNames()
   latLon()
   cityStates()
# ------------------------------------------------------
# CommonCityNames.txt

# read in states and strip newlines
def commonCityNames():
   states = [line.rstrip('\r\n') for line in open('states.txt')]

   # arrays to store info
   linesToSave = []
   citiesToSave = []

   # find matching states in zipcodes.txt & states.txt
   # add entire line to linesToSave if there's a match
   with open('zipcodes.txt') as f:
      for line in f:
         count=0
         for word in line.split('\t'):
            count+=1
            if(count == 5):
               for state in states:
                  if word == state:
                     linesToSave.append(line)

   # for each matching state, get city and add it to citiesToSave
   for l in linesToSave:
      count=0
      for word in l.split('\t'):
         count+=1
         if(count == 4):
            citiesToSave.append(word)
```

```
    # sort citiesToSave and write to file w/ one city per line
    citiesToSave = (sorted(citiesToSave))
    fileToWrite = open('CommonCityNames.txt', 'w')

    for city in citiesToSave:
        fileToWrite.write(city + '\n')


# ---------------------------------------------------------
# LatLon.txt

def latLon():
    # read in zips and strip new lines
    zips = [line.rstrip('\r\n') for line in open('zips.txt')]

    zipsLines = []
    latAndLon = []

    # add info for the zipcodes in zips.txt to zipsLines
    # no duplicate zipcodes are added to zipsLines
    with open('zipcodes.txt') as f:
        for line in f:
            count=0
            for word in line.split('\t'):
                count+=1
                if(count == 2):
                    for zip1 in zips:
                        if word == zip1:
                            if not zipsLines:
                                zipsLines.append(line)
                            # before adding, check for duplicate
                            for line2 in zipsLines:
                                count2=0
                                for word2 in line2.split('\t'):
                                    count2+=1
                                    if(count2 == 2 and word2 != zip1):
                                        zipsLines.append(line)

    # get lat and lon of the zipcode entry and add them to latAndLon
    for l in zipsLines:
        count=0
        for word in l.split('\t'):
            count+=1
            if(count == 7):
                latAndLon.append(word) # lat
            if(count == 8):
                latAndLon.append(word) # lon

    # split the latitudes and longitudes into their own lists
    compList = [latAndLon[x:x+2] for x in range(0, len(latAndLon), 2)]
```

```
    fileToWrite = open('LatLon.txt', 'w')

    # write appropriate info to the LatLon.txt file
    for setList in compList:
        count = 0
        for setList2 in setList:
            count+=1
            fileToWrite.write(setList2)
            if(count == 1):
                fileToWrite.write(' ')
        fileToWrite.write('\n')


# ----------------------------------------------------
# CityStates.txt

def cityStates():
    cities = [line.rstrip('\r\n') for line in open('cities.txt')]
    citiesLines = []

    # # add info for the cities in cities.txt to citiesLines
    # # no duplicate cities are added to citiesLines
    with open('zipcodes.txt') as f:
        for line in f:
            count=0
            for word in line.split('\t'):
                count+=1
                if(count==4):
                    for city1 in cities:
                        city = city1.upper()
                        if(word == city):
                            # if list is empty, add the line
                            if not citiesLines:
                                citiesLines.append(line)
                            else:
                                for line2 in citiesLines:
                                    count2=0
                                    for word2 in line2.split('\t'):
                                        count2+=1
                                        if(count2 == 4):
                                            if(word2 != city):
                                                citiesLines.append(line)

    states = []
    words = []
    fileToWrite = open('CityStates.txt', 'w')
    # for each city, get its states and write to file
    for city3 in cities:
        states = []
        for entry in citiesLines:
            words = []
```

```
        for word3 in entry.split('\t'):
            words.append(word3)
        if(words[3]==city3.upper()):
            if words[4] not in states:
                states.append(words[4])
    # sort states before writing
    states.sort()
    fileToWrite.write('\n' + city3 + ': ')
    for state in states:
        fileToWrite.write(state + ' ')

main()
```

```
  GNU nano 2.2.6                File: CommonCityNames.txt

█BERDEEN
ACME
ADA
ADA
ADA
ADA
ADA
ADAIRO
ADAMS COUNTY
ADAMS MILLS
ADAMSVILLE
ADDISON
ADDISON TOWNSHIP
ADDISON TWP
ADDYSTON
ADELPHI
ADENA
ADRIAN
ADRIAN
AFTON
AHMEEK
AK STEEL
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
AKRON
```

```
  GNU nano 2.2.6                File: LatLon.txt

41.51 -84.3
39.5 -84.74
39.5 -84.74
39.5 -84.74
39.5 -84.74
39.5 -84.74
39.5 -84.74
42.28 -83.17
42.28 -83.17
42.28 -83.17
42.28 -83.17
42.28 -83.17
42.28 -83.17
42.28 -83.17
```

---

sum.java
SoC: All requirements were met for this problem in Java.

```
// Stuart Wyse
// CSE 465 - Term Project
// sum.java

import java.util.ArrayList;
import java.util.Arrays;

public class sum {
	public static String answer; // global String used to check for a valid sum
	public static int count; // used in the sums function to count to number of
                                                        // times we enter that function - before this, a
                                                        // target of 0 would always return true

	/*
	 * Call the test cases in the assignment description. Reset answer to an
	 * empty String after each call, and set count back to 0.
	 */
	public static void main(String args[]) {
		ArrayList<Integer> list = new ArrayList<Integer>();
		list.add(1);
		list.add(2);
		list.add(3);
		list.add(4);
		// list = [1,2,3,4]
		System.out.println(summation(list, 10));
		answer = "";
		count = 0;

		System.out.println(summation(list, 14));
		answer = "";
```

```
		count = 0;

		System.out.println(summation(list, 0));
		answer = "";
		count = 0;

		list.add(0);
		// list = [1,2,3,4,0]

		System.out.println(summation(list, 0));
		answer = "";
		count = 0;

		ArrayList<Integer> list2 = new ArrayList<Integer>();
		list2.add(1);
		list2.add(2);
		list2.add(3);
		list2.add(-4);

		// list2 = [1,2,3,-4]
		System.out.println(summation(list2, -3));
		answer = "";
		count = 0;

		ArrayList<Integer> list3 = new ArrayList<Integer>();
		list3.add(1);
		list3.add(2);
		list3.add(8);
		list3.add(2);

		// list3 = [1,2,8,2]
		System.out.println(summation(list3, 5));
		answer = "";
		count = 0;
}

/*
 * Calls the sums function and returns true or false, depending if the
 * target sum was found.
 */
public static boolean summation(ArrayList<Integer> list, int target) {
		if (list.size() == 0 || list == null) {
			return false;
		}
		if (list.contains(target)) {
			return true;
		}
		String answer1 = sums(list, target, new ArrayList<Integer>());
		if (answer1 != null && answer1.length() > 0) {
			return true;
		} else
```

```
                return false;
        }

        /*
         * Recursive function to find all sums in the arraylist. Returns a String
         * saying if a target sum was found, otherwise the String is null.
         */
        public static String sums(ArrayList<Integer> list, int target, ArrayList<Integer> partialList) {
                count++;
                int s = 0;
                for (int x : partialList)
                        s += x;
                if (count > 1 && s == target) {
                        answer = "YES";
                        return answer;
                }
                for (int i = 0; i < list.size(); i++) {
                        ArrayList<Integer> remaining = new ArrayList<Integer>();
                        int n = list.get(i);
                        for (int j = i + 1; j < list.size(); j++) {
                                remaining.add(list.get(j));
                        }
                        ArrayList<Integer> partial = new ArrayList<Integer>(partialList);
                        partial.add(n);
                        sums(remaining, target, partial);
                }
                return answer;
        }
}
```

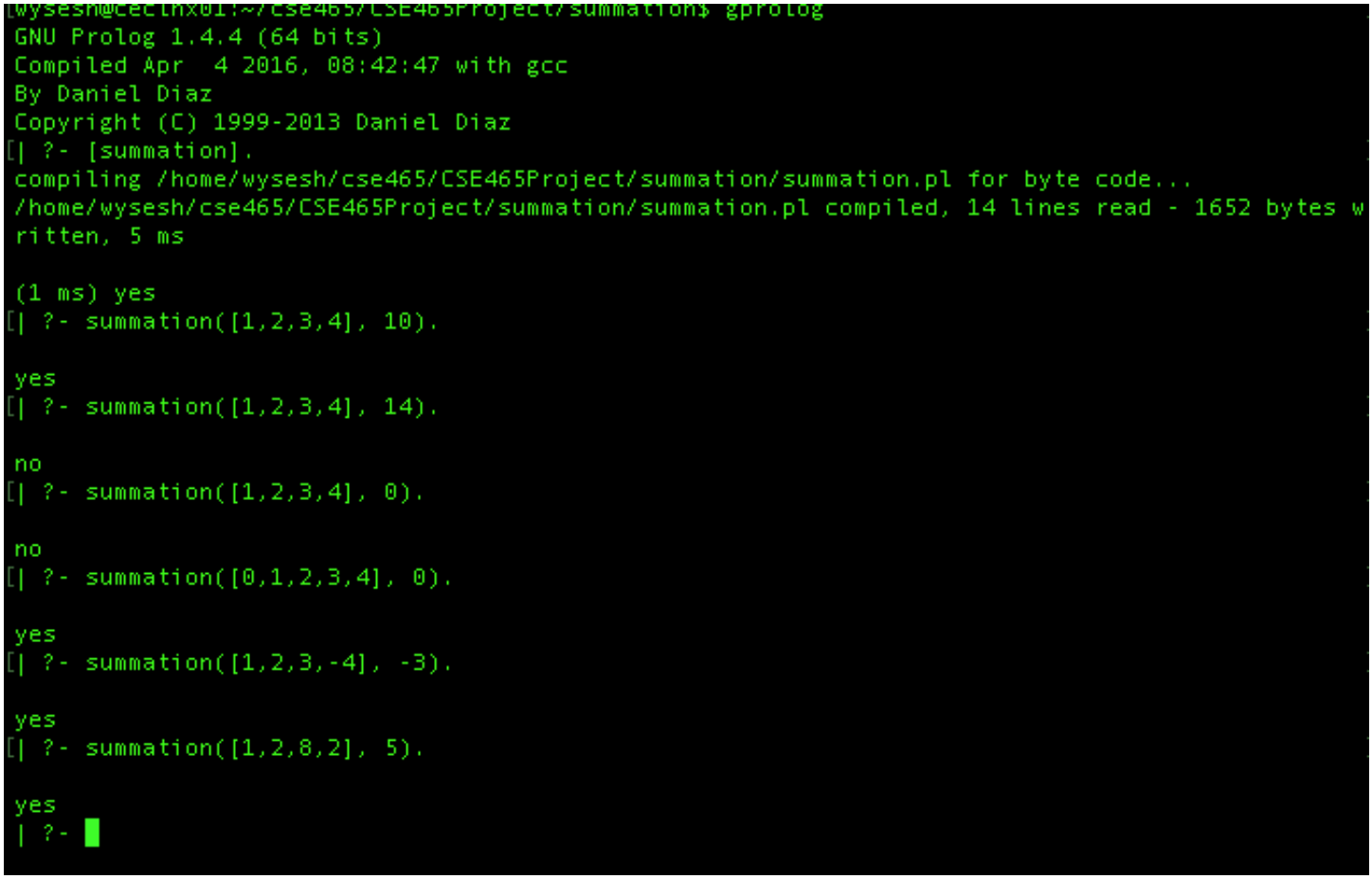
```
  GNU nano 2.2.6                File: sumjava.out

rue
false
false
true
true
true
```

---

summation.pl
SoC: All requirements were met for this problem in Prolog.

```
/**
* Stuart Wyse
* CSE 465 - Term Project
* summation.pl
*/

summation(List, Target) :-  sublst(List, X), sum(X, Target), !.
```

```
sublst([], []).
sublst([E|Tail], [E|NTail]) :- sublst(Tail, NTail).
sublst([_|Tail], NTail) :- sublst(Tail, NTail).

sum([N], N) :- number(N).
sum([H|T], N) :- number(H), sum(T, Y), N is H + Y.
```

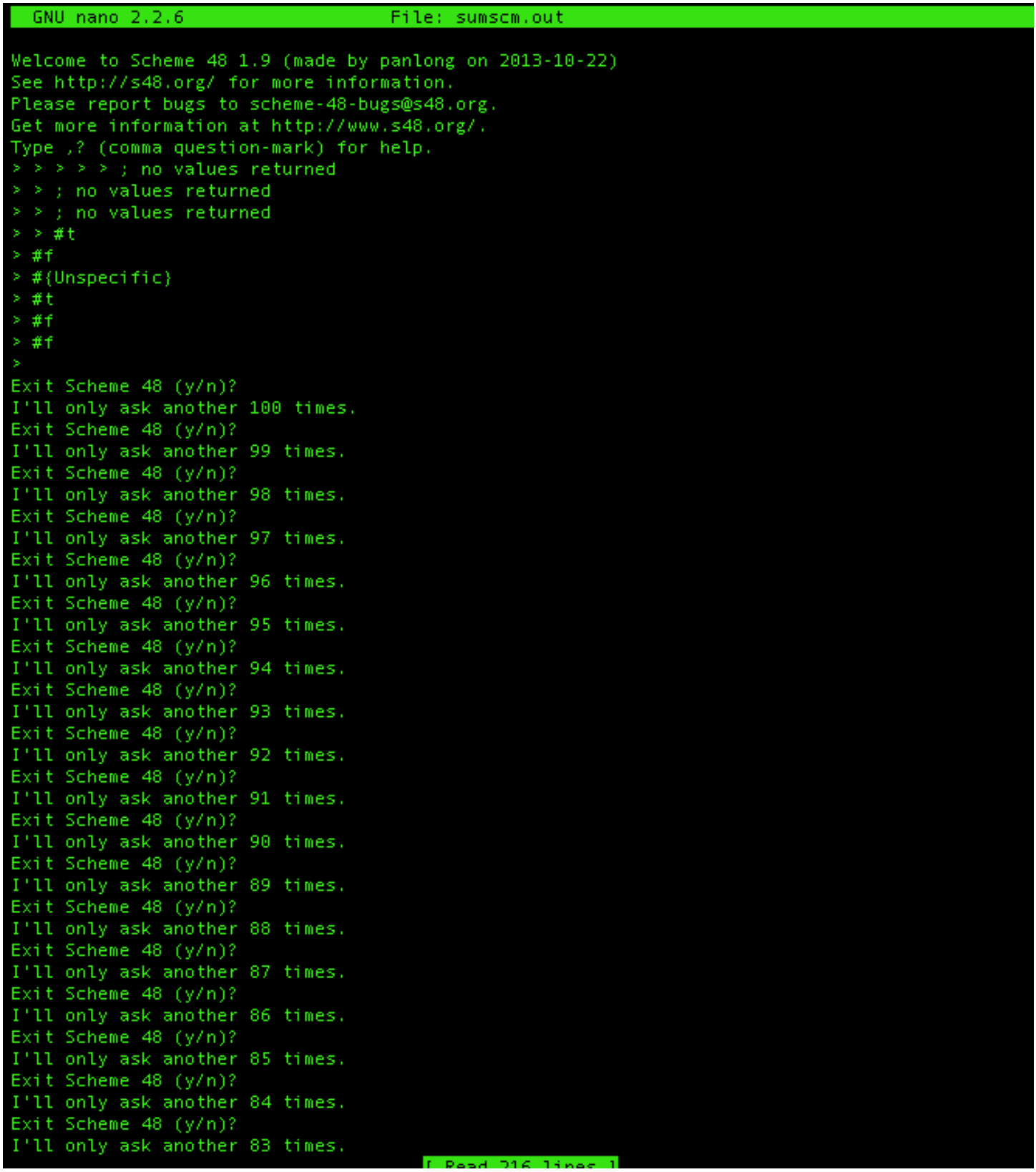


---

sum.scm
SoC: This solution is not completely done. It will only find the target if the target can be found by adding the numbers in the list from left to right. If the target is in the list, it will also be able to find it. When the target is 0 and there should be no solution found, instead of outputting '#f', it will output '(Unspecific)'.

```
; Stuart Wyse
; CSE 465 - Term Project
; sum.scm

(define count 0)

(define (SumListH sum L target count)
  (set! count (+ 1 count)) ; add one to count
  (if (null? L) ; is the list null?
     (if (= target sum) ; if yes, check for target
```

```
        (if (< count 2) #f ; if target = sum & count < 2, return #f
           #t ; if target = sum & count !< 2, return #t
        )
        #f ; list is null and target != sum, return #f
     )
     (if (= target sum) ; list is not null, so check for target
        (if (> count 2) #t) ; target = sum, so check for count > 2 - if yes, return #t
        (SumListH (+ sum (car L)) (cdr L) target count) ; target != sum, so call function ag$
     )
  )
)

(define (SumList L target)
  (if (memq target L) #t ; if target is in list, return #t
     (SumListH 0 L target 0)
  )
)

(SumList '(1 2 3 4) 10)
(SumList '(1 2 3 4) 14)
(SumList '(1 2 3 4) 0)
(SumList '(0 1 2 3 4) 0)
(SumList '(1 2 3 -4) -3)
(SumList '(1 2 8 2) 5)
```



---

zpm.java
SoC: All requirements were met for this problem in Java.

/*

```
 * Stuart Wyse
 * CSE 465 - Term Project
 * zpm.java
 */

import java.util.HashMap;
import java.util.Scanner;
import java.io.File;
import java.util.ArrayList;

public class zpm {
	public static HashMap<String, String> variables = new HashMap<String, String>();
	public static int count = 0;

	public static void main(String[] args) {
		try {
			Scanner in = new Scanner(System.in);
			if (args.length > 0) {
				String fileName = args[0];
				File file = new File(fileName);
				in = new Scanner(file);
			} else {
				System.err.println("ERROR: File name missing.\n");
				System.exit(0);
			}

			// this counter is to show line #s of runtime errors
			count = 0;

			while (in.hasNextLine()) {
				count++;
				String line = in.nextLine();
				parseStatememt(line);
			}

		} catch (Exception e) {
			System.out.println(e);
		}
	}

	// Tokenize a single line of source code, decide what kind of statement it
	// is, and call the appropriate method
	public static void parseStatememt(String line) {
		String[] tokens = line.split("\\s+");
		int numTokens = tokens.length;

		if (numTokens > 4 && tokens[2].contains("\"")) {
			String[] oldTokens = tokens;
			tokens = new String[4];
			tokens[0] = oldTokens[0];
			tokens[1] = oldTokens[1];
```

```
                tokens[2] = line.substring(line.indexOf("\""), line.lastIndexOf("\"") + 1);
                tokens[3] = oldTokens[oldTokens.length - 1];

                numTokens = tokens.length;
        }

        if (numTokens == 3) {
                analyzePrint(tokens);
        } else if (numTokens == 4) {
                analyzeAssignment(tokens);
        } else {
                // If we don't have a blank line, it is a FOR loop
                if (line.length() != 0) {
                        analyzeFor(line);
                }
        }
}

// Tokenize a PRINT statement and print appropriate info to the console
public static void analyzePrint(String[] tokens) {
        // Print statement has 3 tokens
        String toPrint = tokens[1];
        String prefix = "";

        if (findType(toPrint) == 2) {
                if (!variables.containsKey(toPrint)) {
                        runtimeError();
                } else {
                        prefix = toPrint + " =";
                        toPrint = variables.get(toPrint);
                }
        }

        if (findType(toPrint) == 0) {
                toPrint = removeQuotes(toPrint);
        }

        System.out.println(prefix + " " + toPrint);
}

// Tokenize an assignment statement and create or update the appropriate
// variable
public static void analyzeAssignment(String[] tokens) {
        String varName = tokens[0];
        String statement = tokens[1];
        String newValue = tokens[2];

        if (findType(newValue) == 2) {
                // Throw an error if variable doesn't exist
                if (variables.containsKey(newValue)) {
                        newValue = variables.get(newValue);
```

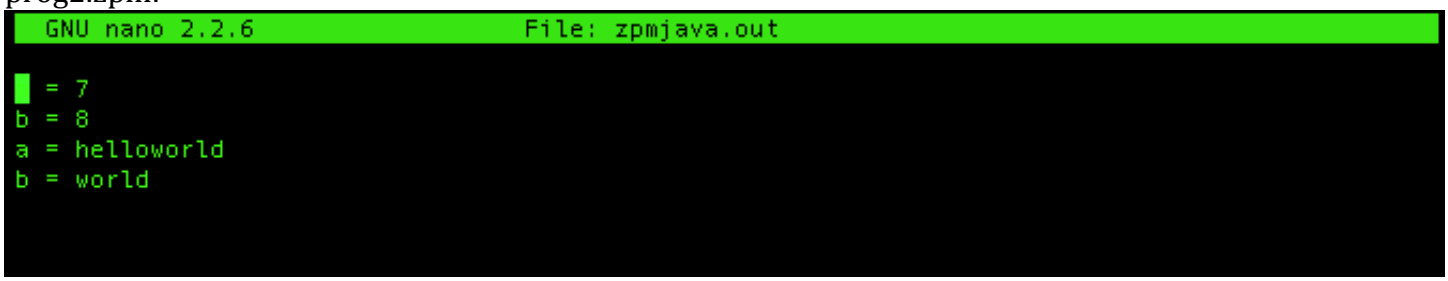
```
                } else {
                        runtimeError();
                }
        }

        int newType = findType(newValue);

        // Make sure types work together if variable is already declared
        if (variables.containsKey(varName)) {
                int oldType = findType(variables.get(varName));
                if (!statement.equals("=") && newType != oldType) {
                        runtimeError();
                }
        }

        // Check to see what kind of statement it is and perform that operation
        switch (statement) {
        case "+=":
                if (newType == 0) {
                        String string1 = removeQuotes(variables.get(varName));
                        String string2 = removeQuotes(newValue);
                        // Put quotes back so we know it's a string
                        newValue = "\"" + string1 + string2 + "\"";
                } else {
                        newValue = (Integer.parseInt(variables.get(varName)) +
Integer.parseInt(newValue)) + "";
                }
                break;
        case "-=":
                // Can't do this w/ strings
                if (newType == 0) {
                        runtimeError();
                } else {
                        newValue = (Integer.parseInt(variables.get(varName)) -
Integer.parseInt(newValue)) + "";
                }
                break;
        case "*=":
                // Can't do this w/ strings
                if (newType == 0) {
                        runtimeError();
                } else {
                        newValue = (Integer.parseInt(variables.get(varName)) *
Integer.parseInt(newValue)) + "";
                }
                break;
        default:
        }
        variables.put(varName, newValue);
    }
```

```
// Tokenize a for loop and complete the appropriate action
public static void analyzeFor(String line) {
	line = line.trim();
	line = line.substring(4, line.length() - 7);

	line = line.trim();
	int spaceAfterNum = line.indexOf(" ");
	int loopCond = Integer.parseInt(line.substring(0, spaceAfterNum));
	line = line.substring(spaceAfterNum);

	for (int i = 0; i < loopCond; i++) {
		executeStatementList(line);
	}
}

// Determine the "type" of a string token
// Meaning of int returned: 0 = string, 1 = int, or 2 = variable name
public static int findType(String toDetermine) {
	// If first character is a ", it's a String
	if (toDetermine.charAt(0) == '\"') {
		// string
		return 0;
	}
	try {
		// int
		Integer.parseInt(toDetermine);
		return 1;
	} catch (Exception e) {
		// variable name
		return 2;
	}
}

// Split a statement list into the different statements and execute each statement
public static void executeStatementList(String line) {
	ArrayList<String> stmts = new ArrayList<String>();

	while (line.length() > 0) {
		line = line.trim();
		if (line.startsWith("FOR")) {
			int endfor = line.lastIndexOf("ENDFOR");
			stmts.add(line.substring(0, endfor + "ENDFOR".length()));
			// everything after the ENDFOR
			line = line.substring(endfor + "ENDFOR".length());
		} else {
			int semicolon = line.indexOf(";");
			stmts.add(line.substring(0, semicolon + 1));
			line = line.substring(semicolon + 1);
		}
	}
```

```
            for (String s : stmts) {
                parseStatememt(s);
            }
        }

        // Remove the the quotes from a string
        public static String removeQuotes(String str) {
            return str.substring(1, str.length() - 1);
        }

        // Print runtime error and exits the program
        public static void runtimeError() {
            System.out.println("RUNTIME ERROR: line " + count);
            System.exit(0);
        }

}
```

prog1.zpm:

prog2.zpm:



prog5.zpm:

---

zpm.cpp

SoC: The only requirement not met for this problem was the handling of FOR loops. I print out a message warning the user of this. I also get a segmentation fault when compiling.

```
// Stuart Wyse
// CSE 465 - Term Project
// zpm.cpp

#include <string>
#include <fstream>
#include <iostream>
#include <cstring>
#include "zpm.h"

using namespace std;

int count = 0;

// Constructor
zpm::zpm() {

}

// Destructor
zpm::~zpm() {
  deletePointers();
}

// Finds the correct funtion to send the line to intepret it
void zpm::parseStatement(std::string line) {
  // if PRINT is not on a line, we know it's an assignment statement
  if(line.find("PRINT") == -1) {
    if(line.find("FOR") != -1) {
      std::cout << "Does not handle FOR loops." << std::endl;
    } else {
    analyzeAssignment(line);
  }
  } else {
    analyzePrint(line);
  }
}

void zpm::parseFile(char* file) {
  std::ifstream zpmFile;
  zpmFile.open(file);

  // parse each statement in the file
  std::string line = "";
  while(getline(zpmFile, line)) {
    lineNumber++;
    parseStatement(line);
  }
```

```
}

void zpm::analyzeAssignment(std::string line) {
  // name of variable
  std::string variable = nextToken(&line);
  // assignment operator
  std::string operator1 = nextToken(&line);
  // value on right side of statement
  std::string val = nextToken(&line);
  Data value;

  Type variableType = findType(val);

  switch(variableType) {
    // Type variable
    case typeVar: {
      // if it doesn't have value, throw error
      if(varTable.find(val) == varTable.end()) {
        runtimeError();
      } else {
        value = varTable[val];
      }
      break;
    }

    // Type string
    case typeString: {
      value.typeFlag = typeString;
      val = trimQuotes(val);
      value.s = new char[val.length() + 1];
      std::strcpy(value.s, val.c_str());
      break;
    }

    // Type int
    case typeInt: {
      value.typeFlag = typeInt;
      value.i = std::stoi(val);
      break;
    }
    // If not one of those types, throw error
    default: {
      runtimeError();
    }
  }

  // Make sure the operator is valid, depending on the variableType
  if(operator1 != "=") {
    if(varTable[variable].typeFlag != value.typeFlag) {
      runtimeError();
    }
```

```
  if((operator1 == "*=" || operator1 == "-=") && value.typeFlag != typeInt) {
    runtimeError();
  }
 }

 // Assign based on the type of assignment
 if(operator1 == "=") {
  varTable[variable] = value;
 } else if(operator1 == "+=") {
  if(value.typeFlag == typeInt) {
   varTable[variable].typeFlag = typeInt;
   varTable[variable].i = varTable[variable].i + value.i;
  } else {
   varTable[variable].typeFlag = typeString;
   int newLength = strlen(varTable[variable].s) + strlen(value.s) + 1;
   char* temp = new char[newLength];

   strcpy(temp, varTable[variable].s);
   strcat(temp, value.s);
   strcpy(varTable[variable].s, temp);
   delete temp;
   temp = NULL;
  }
 } else if(operator1 == "*=") {
  varTable[variable].typeFlag = typeInt;
  varTable[variable].i = varTable[variable].i * value.i;
 } else {
  varTable[variable].typeFlag = typeInt;
  varTable[variable].i = varTable[variable].i - value.i;
 }
}

// Print variable name and value of variable
void zpm::analyzePrint(std::string line) {
 nextToken(&line);
 std::string varName = nextToken(&line);

 if(varTable.find(varName) == varTable.end()) {
  runtimeError();
 } else {
  // Print out variable name and value
  Data value = varTable[varName];
  switch(value.typeFlag) {
   case typeString:
    std::cout << varName << " = " << value.s << std::endl;
    break;

   case typeInt:
    std::cout << varName << " = " << value.i << std::endl;
    break;
```

```
    default:
    // Should never get here
    runtimeError();
   }
  }
}

// Determines type of string
// Options are: string, int, or variable
Type zpm::findType(std::string token) {
  if(token[0] == '\"') {
    return typeString;
  } else if(isdigit(token[0])) {
    return typeInt;
  } else {
    return typeVar;
  }
}

// Gets first token in the string and removes it from the line
std::string zpm::nextToken(std::string* line) {
  // find the first space
  int delimiterIndex = line->find_first_of(" ");
  std::string token = line->substr(0, delimiterIndex);
  token = trimWhitespace(&token);

  *line = line->substr(delimiterIndex, (line->length() - delimiterIndex));
  *line = trimWhitespace(line);

  return token;
}


// Deletes the varTable pointers
// This is used in the destructor and runtimeError function
void zpm::deletePointers() {
  typedef std::map<std::string, Data>::iterator it_type;
  for(it_type iterator = varTable.begin(); iterator != varTable.end(); iterator++) {
    delete iterator->second.s;
    iterator->second.s = NULL;
  }
}

// If there is leading or trailing whitespace on a string, this trims that out
// Got some help from StackOverflow to do this
std::string zpm::trimWhitespace(std::string* str) {
  int start = str->find_first_not_of(" ");
  int end = str->find_last_not_of(" ");
  return str->substr(start, (end-start+1));
}
```

```
// Takes in a string and returns that string, but without the quotes ("") around it
std::string zpm::trimQuotes(std::string str) {
  return str.substr(1, str.length() - 2);
}

// Throw runtime error with line number of error
void zpm::runtimeError() {
  std:cerr << "RUNTIME ERROR: line " <<lineNumber <<std::endl;
  deletePointers();
  exit(1);
}

int main(int argc, char* argv[]) {
  if(argc <= 1) {
    std::cerr << "ERROR: File name missing.\n" << std::endl;
    return -1;
  }

  char* zpmFileName = argv[1];
  zpm* interpreter = new zpm();

  interpreter->parseFile(zpmFileName);

  delete interpreter;
  interpreter = NULL;

  return 0;
}
```

prog1.zpm:

```
  GNU nano 2.2.6                File: zpmcpp.out

  = 4
b = 360
```

prog2.zpm:

```
  GNU nano 2.2.6                File: zpmcpp.out

  = 7
b = 8
a = helloworld
b = world
```

prog5.zpm:

---

bandMatrix.cpp
Soc: Does not compile. I believe all requirements have been met, except that my destructor doesn't do anything.

```
// Stuart Wyse
// CSE 465 - Term Project
// bandMatrix.cpp

#include "bandMatrix.h"

band::band(int size) {
	size = size;
	size1 = (3 * size) - 2;
	bandMatrix.resize(size1);
}

band::band(const band &mat) {
	this->size1 = mat.size1;
	this->size = mat.size1;
	for (int i = 0; i < mat.size1; i++)
	this->bandMatrix[i] = mat.bandMatrix[i];
}

band::band(band& mat2) {
	size = mat2.size;
	size1 = mat2.size1;
	for (int i = 0; i < size1; i++) {
		bandMatrix[i] = mat2.bandMatrix[i];
		mat2.bandMatrix[i] = 0;
	}
	mat2.size = 0;
	mat2.size1 = 0;
	for (int i = 0; i < size1; i++)
	mat2.bandMatrix.pop_back();
}

// destructer
band::~band() {
```

```
}

// set the value of the location passed in as row, col pair
void band::set(int row, int col, double num) {
	if (row <= size|| col <= size) {
		// set the main diagonal
		if (row == col) {
			bandMatrix[row] = num;
		}
		// set the diagonal above the main
		if (row == col - 1) {
			bandMatrix[size + row] = num;
		}
		// set the diagonal below the main
		if (col == row - 1) {
			bandMatrix[(size * 2 - 1) + col] = num;
		}
	} else {
		throw std::runtime_error("Index out of bounds.");
	}
}

// get value at passed in row, col pair
double band::get(int row, int col) {
	// the main diagonal
	if (row == col) {
		return bandMatrix[row];
	}
	// diagonal above the main
	if (row == col - 1) {
		return bandMatrix[size + row];
	}
	// diagonal below the main
	if (col == row - 1) {
		return bandMatrix[(size * 2 - 1) + col];
	} if (row > size || col > size) {
		throw std::runtime_error("Index out of bounds.");
	}
	return 0;
}
```

---

bandMatrix.py
SoC: Nothing implemented for this.

```
# Stuart Wyse
# CSE 465 - Term Project
# bandMatrix.py
```

```
def main():
	words()

def words():
	print ("Nothing implemented for bandMatrix.py")

main()
```