CSCE 420 HOMEWORK 4 (FINAL)
Dr. Daugherity
Due: 11:59 P.M. Tuesday, May 1, 2018

"On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment."

_____ _____
Typed or printed name of student Signature of student

NOTE: Please follow your lab instructor's directions for submitting your assignment through CSNET. ONLY ASSIGNMENTS SUBMITTED TO CSNET WILL BE GRADED! Make a printout of each source file and staple it behind this cover sheet. Sign it and turn it in to Han Wang's mailbox (in the hallway near HRBB 312) by 5:00 P.M. Wednesday. IF YOU DO NOT TURN IN A SIGNED COVER SHEET YOUR WORK WILL NOT BE GRADED!

NOTE: Homework will be graded on compute.cse.tamu.edu, using g++7.2.0 with -std=c++17, or javac and java, or python3.6 (not python or python2 or python3).

You are free to develop your programs on any other platform, but it is your responsibility to make sure your programs also compile and execute correctly on compute.cse.tamu.edu as specified.

NOTE: Each file submitted (hw4pr1.cpp, etc.--see below) must begin as follows:
```
//Your name
//Your UIN
//CSCE 420
//Due: May 1, 2018
//hw4pr1.cpp (or whatever this file name is)
```

NOTE: Also write a README.txt file with whatever information is needed to compile and run your programs. Zip the README.txt and the homework files into a single file named hw4.zip and submit to CSNET.

The grade for this lab will be based on style (formatting, variable names, comments, etc.), syntax (no compilation or link errors), and correctness (passes all test cases). Your grade for this lab is:

Problem #	1	2	3	4
Style	/2	/4	/4	/2
Syntax	/3	/6	/6	/3
Correctness	/5	/10	/10	/5
Total	/10	/20	/20	/10

Grand total _____/50

/***/

```
//Name: Matthew Stevens
//UIN: 924000693
//CSCE 420
//Due: May 1, 2018
//hw4pr1.cpp

#include <iostream>


float fuzzyAnd(float x, float y){
	float temp;
	temp = std::min(x, y);
	return temp;
}
//a -> b = not(x and not(y))
float fuzzyImplies(float x, float y){
	float notResult;
	float notY;

	notY = 1.0 - y;
	notResult = 1.0 - fuzzyAnd(x,notY);

	return notResult;
}

int main(int argc, char* argv[]){
	float arrX[] = {0, 0.25, 0.5, 0.75, 1};
	float arrY[] = {0, 0.25, 0.5, 0.75, 1};
	float result;
	result = 0.0;

	printf("The truth table for fuzzyImples: \n");
	printf("x\t   \t y\t  \t result\n");
	for(int i = 0; i < 5; i++){
		for(int j = 0; j < 5; j++){
			result = fuzzyImplies(arrX[i], arrY[j]);
			//x
			printf("%f\t", arrX[i]);
			printf("%f\t", arrY[j]);
			printf("%f\t", result);
			printf("\n");

		}
	}

}
```

```
/*****************************************************************************/
```

```
//Name: Matthew Stevens
//UIN: 924000693
//CSCE 420
//Due: May 1, 2018
//hw4pr2.cpp



#include <iostream>
#include <fstream>
#include <vector>


int main(){
	std::vector<std::pair<std::string, std::string> > compileOrder;
	std::vector<std::pair<std::string, std::string> > sortedOrder;
	std::vector<std::pair<std::string, std::string> > temp;
	std::vector<std::string> completeOrder;
	std::string first;
	std::string second;

	printf("please enter an acyclic path!\n");
	printf("enter the compiler package (first word): ");
	int numRootChild;
	numRootChild = 0;

	//obtain input and set dependencies
	while(std::getline(std::cin,first)){
		printf("enter the package it is dependent on (second word): ");
		std::getline(std::cin,second);

		//temp pair
		std::pair<std::string, std::string> p;
		p = std::make_pair(first, second);
		compileOrder.push_back(p);

		printf("enter the compiler package (first word): ");
	}

	std::cout << "Done receiving input." << std::endl;
	std::cout << "Beginning Dependency sorting" << std::endl;

	for(int i = 0; i < compileOrder.size(); i++){
		std::string root;
		bool isRoot;

		isRoot = true;
		root = compileOrder[i].first;
		for(int j = 0; j < compileOrder.size(); j++){
			if(root == compileOrder[j].second){
				isRoot = false;
			}
			else{

			}
		}

		if(isRoot){
```

```
            //root becomes first element
            sortedOrder.push_back(compileOrder[i]);
            numRootChild++;
      }
}

//begin with root
std::string current;
current = sortedOrder[0].first;
int children;
int position;
position = 0;
//set root dependencies in order
while(sortedOrder.size() != compileOrder.size()){
      children = 0;
      //for the input size
      for(int i = 0; i < compileOrder.size(); i++){
            //find current node
            if(compileOrder[i].first == current){
                  bool independent;
                  independent = true;
                  //find all the seconds that are dependent on something else
                  for(int j = 0; j < compileOrder.size(); j++){
                        if(compileOrder[i].second == compileOrder[j].first){
                              independent = false;
                              children++;
                              sortedOrder.push_back(compileOrder[j]);
                        }
                  }
                  if(independent){
                        children++;
                  }

            }
      }
      position = position + children;

      current = sortedOrder[position].first;

}

//remove duplicates
for(int i = sortedOrder.size()-1; i >= 0; i--){
      bool contains;
      contains = false;
      if(completeOrder.empty()){
            completeOrder.push_back(sortedOrder[i].second);
      }
      for(int j = 0; j < completeOrder.size(); j++){
            if(completeOrder[j] == sortedOrder[i].second){
                  contains = true;
            }
      }

      if(!contains){
            completeOrder.push_back(sortedOrder[i].second);
      }
}
```

```
	std::cout << "done removing duplicates" << std::endl;
	std::cout << "Compile in numerical order below: " << std::endl;

	for(int i = 0; i < completeOrder.size(); i++){
		std::cout << i+1 << ". " << completeOrder[i] << std::endl;
	}
	std::cout << "Lastly, compile " << sortedOrder[0].first << std::endl;

	return 0;
}
/*****************************************************************************/
//Name: Matthew Stevens
//UIN: 924000693
//CSCE 420
//Due: May 1, 2018
//hw4pr3.cpp

#include <iostream>
#include <fstream>
#include <vector>

int main(){

	std::vector<int> counters;
	char alpha[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
	std::string gram;
	std::string dictionary;
	int max;
	max = 0;
	for(int i = 0; i < 26; i++){
		counters.push_back(0);
	}

	printf("enter a n-gram: ");

	printf("\n");

	while(std::getline(std::cin,gram) ){
		std::cout << "you have entered: " << gram << std::endl;

		std::ifstream ifs;
		ifs.open("/usr/share/dict/words", std::ifstream::in);
		//begin library analysis of n-gram
		while(std::getline(ifs,dictionary)){
			if(dictionary.size() > gram.size()){
				for(int i = 0; i < (dictionary.size() - gram.size()); i++){
					if(dictionary[i] == gram[0] && (dictionary.size()-i)
> gram.size()){
						for(int j = 1; j < gram.size(); j++){
							if(dictionary[i+j] != gram[j]){
								break;
							}
							//end of the gram
							else if(j == gram.size()-1){
								char c;
								c = dictionary[i+j+1];
								//52 letters (caps and lowercase)
								for(int m = 0; m < 52; m++){
									if(c == alpha[m]){
```

```
                                                                  if(m > 25){
                                                                        int num;
                                                                        num = m % 26;
                                                                        counters[num] +=
1;
                                                                  }
                                                                  else{
                                                                        counters[m] += 1;
                                                                  }
                                                            }
                                                      }
                                                }
                                                else{

                                                }
                                          }
                                    }
                              }
                        }

            }
            ifs.close();
            //find max
            for(int i = 0; i < counters.size(); i++){
                  if(counters[i] > max){
                        max = counters[i];
                  }
            }

            for(int i = 0; i < counters.size(); i++){
                  if(counters[i] == max){
                        std::cout << "the highest occurring letter after the n-gram
is: " << alpha[i] << " with " << max << " occurances " << std::endl;
                  }
                  else if(max == 0){
                        std::cout << "there are no results that contain: " << gram
<< std::endl;
                  }
                  else{

                  }
            }

            //reset counter
            for(int i = 0; i < counters.size(); i++){
                  counters[i] = 0;
            }

            printf("enter a new n-gram or ctrl+d to quit: ");
      }
      return 0;
}
```

```
/*************************************************************************/
```

//Name: Matthew Stevens
//UIN: 924000693
//CSCE 420
//Due: May 1, 2018
//hw4pr4.txt

a. Nearest distance is 16 meters, what is the largest disparity (in pixels)?
distance from cameras midpoint to object, z = 16m
width of sensor, w = 10 cm
distance between cameras, b = 1
focal length = 16cm
distance from y = 0 to sensor within camera = .5cm
disparity = ((2*pixels)/width of sensor in cm) * (distance between sensors and focal lense)/(distance between sensors in cm) * (distance from y = 0 to camera lense / distance from cameras to objects in meters)

disparity = ((2*512)/ 10)*(16/1)*(.5/16)
disparity = 51.2 => 51 pixels

b. What is the distance resolution at 16 meters, due to the pixel spacing?
From the disparity from a,
disparity a = 51 pixels
disparity b = 52 pixels

51 = ((2*pixels)/width of sensor in cm) * (distance between sensors and focal lense)/(distance between sensors in cm) * (distance from y = 0 to camera lense / distance from cameras to objects in meters)

51 = ((2*512))/10) * (Z)/(1)*(.5/16)
Z = 15.9375

52 = ((2*pixels)/width of sensor in cm) * (distance between sensors and focal lense)/(distance between sensors in cm) * (distance from y = 0 to camera lense / distance from cameras to objects in meters)

52 = ((2*512))/10) * (Y)/(1)*(.5/16)
Y = 16.25

Resolution = |Y - Z| = 16.25 - 15.9375 = .3125m = 31.25 cm

c. What distance corresponds to a disparity of one pixel?

1 pixel = ((2*pixels)/width of sensor in cm) * (distance between sensors and focal lense)/(distance between sensors in cm) * (distance from y = 0 to camera lense / distance from cameras to objects in meters)

1 pixel = ((2*512pixels)/10cm) * (16m) / (1m) * (.5/z)
z = 819.2m