**CS 350:COMPUTER SYSTEM CONCEPTS Kharon Harrell**

**PROGRAMMING ASSIGNMENT 1**

**Q1. Pythagorean Triples**

**Problem Statement**

A Pythagorean triple consists of three positive integers *a, b,* and *c* such that . Every right triangle has sides that satisfy this formula. For this assignment, we are tasked with finding all the Pythagorean triples no larger than 500.

**Problem Analysis**

This program is expected to identify all Pythagorean triples under 500.

Input(s): integers

Output(s): 3 integers that satisfy the formula

Constraint(s): integers < 500

**Algorithm Design**

Initial Algorithm

1. Create 3 nested for loops to iterate for each variable *a, b,* and *c*
2. Test all combinations up to 500
3. Display Pythagorean triples

**Test Plan**

Three test cases is sufficient; when *a=b=c=1,* common Pythagorean triples, and uncommon. For each variable *a, b,* and *c,* there will be a corresponding nested for loop. This will run through each for loop for all positive possibilities. If there is a Pythagorean pair, I will store the integer values and display them.

**Implementation**

**// -------------------------------------------------------------**

**// Q1.c**

**// Program that identifies all Pythagorean triples up to 500**

**// Author: Kharon Harrell**

**// CS350-001**

**// Instructor: Don Adjeroh**

**// Last Modified: Oct. 1, 2019**

**// ------------------------------------------------------------**

**#include <stdio.h>**

**#include <math.h>**

**#define N 500**

**int sqrSum(int a, int b);**

**int main()**

**{**

**int sum, cSum = 0;**

**int numTrip = 0;**

**printf("Pythagorean Triples:\n");**

**for(int a=1; a<N; a++) //starts at one**

**{**

**for(int b=a+1; b <N; b++) //starts one greater than a to prevent duplicates**

**{**

**sum = sqrSum(a, b);**

**for(int c=b; c <= sum; c++) // starts at b and ends at the sum of the squares to prevent unnneccesary computation**

**{**

**cSum = pow((double)c, 2);**

**if(sum==cSum)**

**{**

**numTrip++; //keeps track of the # of Pythagorean triples**

**printf("%d: %d %d %d\n", numTrip, a, b, c);**

**}**

**}**

**}**

**}**

**return 0;**

**}**

**int sqrSum(int a, int b)**

**{**

**double sqA = (pow((double)a, 2));**

**double sqB = (pow((double)b, 2));**

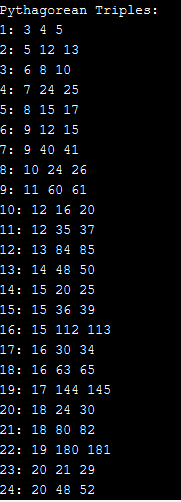
**int sum = sqA + sqB;**

**return sum;**

**}**

**Testing**

After successful compilation, I realized the program ran on it’s own without the need for input. My test cases were all numbers up to 500.

****

**Q2. Temperature in Celsius and Fahrenheit**

**Problem Statement**

For this program, we will convert Celsius to Fahrenheit and vice versa, keeping track of all the Fahrenheit equivalents from 0 to 100 degrees and all of the Celsius equivalents from 32 to 312 degrees.

**Problem Analysis**

This program will convert Fahrenheit to Celius and vice versa and store all of the integers that are equivalent.

Input(s): 2 integers

Output(s): equivalent integers

Constraint(s): Fahrenheit values 0-100, Celsius values 32-212

Formulas:

**Algorithm Design**

Initial Algorithm

1. Create two for loops; one for Fahrenheit values 0-100 and Celsius values 32-212
2. Test Fahrenheit values with formula
3. Test Celsius values with formula
4. Store all values when
5. Display all values in a tabular format

**Test Plan**

For both Fahrenheit and Celius, there will be two separate for loops that test the corresponding values for each. Each conversion will be stored and compared to each value for equivalence. In this scenario, two test cases should suffice; values below/above the scope and values that fall within the scope.

**Implementation**

**// -------------------------------------------------------------**

**// Q2.c**

**// Program that converts F->C & C->F**

**// Author: Kharon Harrell**

**// CS350-001**

**// Instructor: Don Adjeroh**

**// Last Modified: Oct. 1, 2019**

**// ------------------------------------------------------------**

**#include <stdio.h>**

**#include <math.h>**

**int celsius(int fahrenheit);**

**int fahrenheit(int celsius);**

**void printFahrenheit();**

**void printCelsius();**

**int main()**

**{**

**printFahrenheit();**

**printCelsius();**

**}**

**//converts fahrenheit to celsius**

**int celsius(int f)**

**{**

**return ((f-32)\*((float)5/9));**

**}**

**//converts celsius to fahrenheit**

**int fahrenheit(int c)**

**{**

**return (c\*((float)9/5))+32;**

**}**

**void printFahrenheit()**

**{**

**printf("Fahrenheit -> Celsius\n");**

**for(int i=0; i <=100; i++)**

**{**

**int cels = celsius(i);**

**printf("%3d %3d\n", i, cels);**

**}**

**}**

**void printCelsius()**

**{**

**printf("Celsius -> Fahrenheit\n");**

**for(int i=32; i <=212; i++)**

**{**

**int fahr = fahrenheit(i);**

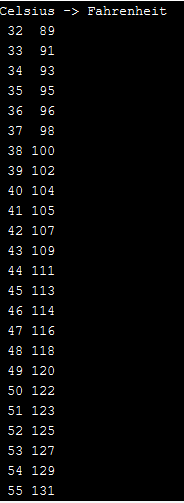
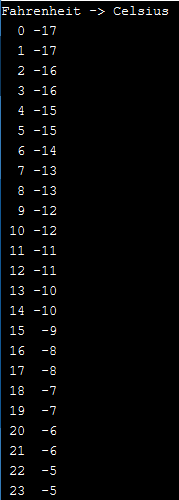
**printf("%3d %3d\n", i, fahr);**

**}**

**}**

**Testing**

After successful compilation, I validated all of the calculations and was able to determine the program ran successfully and successfully printed the correct equivalents.



**Q3a. Greatest Common Divider with iteration**

**Problem Statement**

For this function, we will test two numbers by finding the greatest integer that evenly divides in each of the numbers and returns the results.

**Problem Analysis**

I am tasked with finding the greatest common divisor of two integers.

Input(s): 2 integers;

Output(s): the greatest integer that evenly divides between the two

Formula:

Constraint(s):

**Algorithm Design**

Initial Algorithm

1. Read in two integers
2. Create a for loop that that acts as the GCD to iterate with
3. Take the modulus of both integers and store largest value that both divide evenly into
4. Display result

**Test Plan**

We will take the modulus of each number, each iteration until the largest number prevails. Proof of concept requires three test cases; negative integers, and a positive integers.

**Implementation**

**// -------------------------------------------------------------**

**// Q3A.c**

**// Program that find the GCD between two numbers iteratively**

**// Author: Kharon Harrell**

**// CS350-001**

**// Instructor: Don Adjeroh**

**// Last Modified: Oct. 1, 2019**

**// ------------------------------------------------------------**

**#include <stdio.h>**

**int gcd(int a, int b);**

**int main()**

**{**

**gcd(1, 1);**

**gcd(50, 10);**

**gcd(24, 36);**

**return 0;**

**}**

**int gcd(int a, int b)**

**{**

**int gcd = 1;**

**for(int i=2; i <= a && i <= b; i++)**

**{**

**if(a%i==0 & b%i==0)**

**{**

**gcd = i;**

**}**

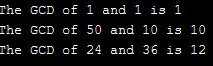
**}**

**printf("The GCD of %d and %d is %d\n", a, b, gcd);**

**return gcd;**

**}**

**Testing**

****

**Q3b. Greatest Common Divider with recursion**

**Problem Statement**

For this function, we will test two numbers by finding the greatest integer that evenly divides in each of the numbers recursively and returns the results.

**Problem Analysis**

I am tasked with finding the greatest common divisor of two integers.

Input(s): 2 integers an

Output(s): the greatest integer that evenly divides between the two

Formula: If , then . Else,

**Algorithm Design**

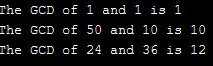
Initial Algorithm

1. Read in two integers and
2. Create the base case for when
3. Create the recursive for all other cases
4. Display result

**Test Plan**

For this function, each test cases would only run as many times as the smallest number, at max. We will take the modulus of each number recursively for three test cases; negative integers, and a positive integers.

**Testing**

****

**Implementation**

**// -------------------------------------------------------------**

**// Q3B.c**

**// Program that find the GCD between two numbers recursively**

**// Author: Kharon Harrell**

**// CS350-001**

**// Instructor: Don Adjeroh**

**// Last Modified: Oct. 1, 2019**

**// ------------------------------------------------------------**

**#include <stdio.h>**

**int gcd(int a, int b);**

**int main()**

**{**

**printf("The GCD of %d and %d is %d\n", 1, 1, gcd);**

**printf("The GCD of %d and %d is %d\n", 50, 10, gcd);**

**printf("The GCD of %d and %d is %d\n", 24, 36, gcd);**

**return 0;**

**}**

**int gcd(int a, int b)**

**{**

**//base case**

**if(a==0)**

**{**

**return a;**

**} else if(b==0) {**

**return b;**

**} else if (a==b) {**

**return a;**

**}**

**//recursive case**

**if(a>b)**

**{**

**return gcd(a-b, b);**

**}**

**return gcd(a, b-a);**

**}**

**Q4. The Knight’s Tour Problem**

**Problem Statement**

Tour a chess board using **only** a Knight by stepping on each square at least **once**. A full tour is considered when all squares on a chess board are visited.

**Problem Analysis**

I will need to visit each position on the chess board using the rules of the Knight.

Input(s): integer starting position of Knight

Output(s): full chess board with integers of traveled positions

Constraint(s): Knight **must** be inside chess board, Knight can’t visit the same position twice, Knight must obey rules of the night

**Algorithm Design**

Initial Algorithm

1. While(valid moves) *\*valid moves- a move within constraints*
2. Make a move
3. Print the board

**Test Plan**

In essence, this program will need to simulate a chess board and the rules of the Knight chess piece. Although this program is complex, the test cases are straightforward. There will be three test cases; values that fall within the chess board and illegal values.

**Implementation**

**// -------------------------------------------------------------**

**// Q4.c**

**// Program that simulates all possible full tours of all possible positions //for a Knight**

**// Author: Kharon Harrell**

**// CS350-001**

**// Instructor: Don Adjeroh**

**// Last Modified: Oct. 1, 2019**

**// ------------------------------------------------------------**

**#include <stdio.h>**

**#include <string.h>**

**#include <unistd.h>**

**void printChessBoard(int board[8][8]);**

**int isValid(int row, int col);**

**void fillTable(int table[8][8], int board[8][8]);**

**int nextMove(int row, int col, int board[8][8], int table[8][8], int \*nextR, int \*nextC);**

**int horizontal[8] = {2, 1, -1, -2, -2, -1, 1, 2};**

**int vertical[8] = {-1, -2, -2, -1, 1, 2, 2, 1};**

**int main()**

**{**

**int board[8][8] = {0};**

**int table[8][8] = {0};**

**printChessBoard(board);**

**printf("\n");**

**fillTable(table, board);**

**printChessBoard(table);**

**int row = 0;**

**int col = 0;**

**int fullTours = 0;**

**//touring from every possible cell**

**for (row = 0; row < 8; row++) {**

**for (col = 0; col < 8; col++) {**

**int moveCounter = 1;**

**int nextR = 0, nextC = 0;**

**int r = row;**

**int c = col;**

**//clear the board**

**memset(board, 0, 8\*8\*sizeof(int));**

**board[row][col] = moveCounter;**

**fillTable(table, board);**

**//touring the board while their are possible moves**

**while (nextMove(r, c, board, table, &nextR, &nextC) < 9) {**

**r = nextR;**

**c = nextC;**

**board[r][c] = ++moveCounter;**

**fillTable(table, board);**

**}**

**//keeps track of full tours**

**if (moveCounter == 64) {**

**fullTours++;**

**}**

**}**

**}**

**//clears chess board**

**printChessBoard(board);**

**printf("\n");**

**printChessBoard(table);**

**printf("There were %d full tours out of 64\n", fullTours);**

**return 0;**

**}**

**//print chess board**

**void printChessBoard(int board[8][8])**

**{**

**for(int i=0; i < 8; i++)**

**{**

**for(int j=0; j < 8; j++)**

**{**

**printf("%3d ", board[i][j]);**

**}**

**printf("\n");**

**}**

**}**

**//filling the accessibility table with accessible positions**

**void fillTable(int table[8][8], int board[8][8])**

**{**

**for(int i=0; i<8;i++)**

**{**

**for(int j=0; j< 8; j++)**

**{**

**int access = 0;**

**//counts # of accessible positions**

**for(int count=0; count < 8; count++)**

**{**

**int row = i+vertical[count];**

**int col = j+horizontal[count];**

**if(isValid(row, col) && board[row][col]==0)**

**{**

**access++;**

**}**

**}**

**table[i][j] = access;**

**}**

**}**

**}**

**//checks for valid moves**

**int isValid(int row, int col)**

**{**

**if(row >= 0 && row < 8 && col >= 0 && col < 8)**

**{**

**return 1;**

**}**

**return 0;**

**}**

**//finds the next move by using the accessibility heuristic**

**int nextMove(int row, int col, int board[8][8], int table[8][8], int \*nextR, int \*nextC) {**

**int min = 9;**

**int minR, minC;**

**//finds the cell with lowest accessibility**

**for(int count=0; count < 8; count++)**

**{**

**int r = row+vertical[count];**

**int c = col+horizontal[count];**

**if(isValid(r, c) && board[r][c]==0)**

**{**

**if(table[r][c] < min)**

**{**

**min = table[r][c];**

**minR = r;**

**minC = c;**

**}**

**}**

**}**

**//returns the next move**

**\*nextR = minR;**

**\*nextC = minC;**

**return min;**

**}**

**Testing**

For this program, I decided to generate all possible tours for all possible starting positions. It compiled successfully and looks really cool to run as well.

