Fall 2013 COP 3223: Introduction to Programming
Program 8 (*Is the Air Running?)*

Due Date: *Please Consult WebCourses*

Objective
To give students practice using arrays, reading from files and writing to files.

The Problem
A client is interested in exactly what percentage of the time her air conditioning is running. One way to measure this is to keep a temperature logging device right by an AC vent. The device measures the temperature every 30 seconds. If the temperature drops at least .5 degrees Fahrenheit between readings, it's a good bet that the AC unit just turned on. Likewise, if the temperature rises at least .5 degrees Fahrenheit between readings it's likely that the AC unit just turned off. (For example, if the reading at 1:30:00pm was 74.7 degrees F and the reading at 1:30:30pm was 74.1 degrees F, then we would assume for the 30 seconds starting at 1:30:00pm and ending at 1:30:30pm, the AC was on. If there is no big rise or fall in a 30 second interval, we assume the state of the AC was whatever it was in the previous 30 second interval. *We will assume that the AC is always off at midnight*.

You will be given the temperature readings from one logging device for a whole day (24 hours). Using this information, you must calculate the percentage of time the air was on for the day and produce a bar graph with hourly data.

Input File Format (temp.txt)
This file will have exactly 2881 doubles, one per line, indicating readings of temperatures (in Fahrenheit) starting at midnight and ending at midnight the next day. The readings are taken every 30 seconds. The start of a file might look like this:

```
78.6
78.6
78.5
78.5
```

The Output (temp.out)
First, output a line with the following format:

```
The AC was running X percent of the time.
```

where X is a percentage rounded to 2 decimal places, corresponding to how often the AC was running throughout the whole day.

After this line of output, skip a blank line and output a bar graph with a similar format to the one in the posted file.

Each bar corresponds to the percent of time the AC was running during that hour. Since the bars are shown with 5% increments, only display a star if the percentage for that hour equals or exceeds the marked percentage. Here's the basic idea in how to create it.

First, store the data in an array of size 24, where each entry indicates the percentage of time the AC was running in that hour. (For example, the entry in index 0 stands for the amount of time the AC was running from midnight to 1 am.)

Now, print each row in order, first the row for 100%, then 95%, then 90%, etc. all the way down to 5%. While you are printing a row, for each column, you must ask yourself the question, "Should I print a star at this location, or a space?" A star gets printed if for that hour, the percentage was equal to or greater than that of the row that is currently getting printed. A space gets printed otherwise. Consider the following example:

If the array stored 66% in index 0, 40% in index 1 and 53% in index 2, then imagine printing the row that corresponds to 50%:

You would print a STAR for the first character, since $66 \geq 50$.
You would print a SPACE for the second character, since $40 < 50$.
You would print a STAR for the third character, since $53 \geq 50$.

Program Details

Please read your input from the file "temp.txt" and write your output to the file "temp.out". (Thus, you should use both fscanf and fprintf and no scanf or printf. Namely, when your program executes, it should immediately run, not waiting for any user input.)

References

Textbook: Chapters 11 Notes: Array Lectures

Output Sample

Posted as separate attachments.

Deliverables

You must submit your solution to the problem, *aircond.c*, over WebCourses.

Restrictions

Although you may use other compilers, your program must compile and run using Code::Blocks with gcc. Each of your three programs should include a header comment with the following information: your name, course number, section number, assignment title, and date. Also, make sure you include comments throughout your code describing the major steps in solving the problem.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness
2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.
3) Compatibility to Code::Blocks (in Windows). If your program does not compile in this environment, **the maximum credit you will receive is 50%.**

COP 3223 Fall 2013 Grading Criteria Program 8
Total Points: 100

Execution Points (50 pts) – 25 pts per test case

Breakdown for points for each test case:

1) Calculates correct percentage at the beginning of the output (5 pts) – may give partial if you can see that some part of the calculation is correct and another part isn't.

2) Prints out a valid bar for the first hour. (5 pts)
3) Prints out a valid bar for all the hours. (5 pts)
4) Each bar is correct (10 pts total)

Note: if a program doesn't compile, but all the logic is correct, you may award up to 25 of the 50 execution points. If a simple fix will get the program to compile, just take points off for this, fix it and then grade. This will be left up to your mercy and discretion. Also, if none of the test cases work because of a simple logical error, feel free to deduct 2 to 5 points for that logical error (based on your assessment of its severity) instead of the full 50 points for the test cases.

Points for Code (35 pts)
1) Uses #defines/const (2 pt) (reasonable choices)
2) Opens the input file (3 pt)
3) Reads in exactly 2881 values (5 pt)
4) Has an array of size 24 (and/or 2880) to store appropriate values (5 pts)
5) Has a mechanism to calculate # of segments on out of 120 in an hour (3 pts)
6) Tries to calculate a corresponding percentage (2 pts)
7) Has a print out for the total run-time percentage (doesn't have to be correct) (4 pts)
8) Loops through each row of the chart (3 pts)
9) Prints out a row header (2 pts)
10) Has logic to print out either a space or a star for each slot in a row (5 pts)
11) Prints the bottom of the chart (1 pt)

Points for Comments & Style (15 pts)
1) Header comment with name, course number, section number, assignment title, and date. (4 pts)
2) Comments within code (5 pt)
3) Indentation and use of white space (3 pt)
4) Appropriate variable names (3 pt)

When you give the students comments, just tell them how much you took off and for what. Also, include your initials at the end of your comment so students know who graded their assignment. Don't assign fractional points.

Introduction to C - Program 9
Use of Functions: Casino

Objective
To give students practice in writing functions and calling those functions to perform more complex tasks.

The Problem: Casino
You will write a program that simulates a casino for a single player. The user will initially start with $1000. The user will then be able to choose from the following options:

1) Buy chips
2) Sell chips
3) Play Craps
4) Arup's Game of Dice
5) Status Report
6) Quit

Your program will execute each choice until the quits. At this point all of their chips automatically get sold back to the casino and a message prints out how much money the user has left (of the $1000) after gambling.

Note: Due to the length of this assignment, you will be given a framework (casino_framework.c) to fill in to complete it. The framework will contain each function broken down, a completed main and a completed implementation of Arup's Game of Dice. You will be responsible to fill in the rest of the framework. If you prefer to write the assignment completely on your own, you are welcome to do so, but you must still implement the functions that are required as they are stated in this write up.

Craps
One of the most "fair" games to play at a casino is Craps. Here is one version of how to play:

1) Roll a pair of fair six-sided dice.
2) If you roll a 7 or 11, you win!
3) If you roll a 2, 3, or 12, you lose.
4) Otherwise, record what you've rolled. Let this sum be k; also known as your point.
5) If you rolled a point, continue rolling the pair of dice until you get either your point (k) or a sum of seven on the two dice.
6) If k comes up first, you win!
7) If 7 comes up first, you lose.

Arup's Game of Dice

Amazingly, this game is even more "fair" than Craps, but the house still has a 50.2% chance of winning, which is why the casino hasn't gone broke yet! Here are the rules:

1) Roll a pair of dice.
2) If you roll a sum of 11 or 12, you win.
3) If you roll a sum of 2, you lose.
4) Otherwise, record what you've rolled. Let this sum be k; also known as your point.
5) Roll one more time. If this roll exceeds your point(k), you win!
6) If this roll is the same as your point(k) or lower, you lose.

Buying Chips

Chips cost $11. Whenever a customer buys chips, he/she must give the banker some money. The banker will always give the user the maximum number of chips they can buy with the money given to them and return the leftover cash. You will write a single function that takes care of this transaction.

Selling Chips

The casino buys chips back at $10 a piece. You will write a single function that takes care of this transaction.

Functions you must write

Though you may write more functions, here are function prototypes for the ones you are required to write:

```
// Precondition: None.
// Postcondition: Returns the sum of two random dice rolls.
int pairofdice();

// Precondition: None.
// Postcondition: Plays one game of Craps and returns 1 if
//                the player won and 0 if they lost.
int craps();

// Precondition: None.
// Postcondition: Plays one game of Arup's game of dice and
//                returns 1 if the player won and 0 if they
//                lost.
int arupsdice();
```

```
// Precondition: cash is the address of the variable
//               storing the amount of money the user is
//               wants to spend on chips.
// Postcondition: The number of chips purchased is returned
//                and the variable storing the amount of
//                money the user paid for chips is adjusted
//                to equal the change left over after the
//                transaction.
int buychips(int *cash);

// Preconditions: numchips > 0.
// Postconditions: Returns the cash obtained for selling
//                 numchips number of chips.
int sellchips(int numchips);

// Precondition: The first parameter is the number of
//               chips the user has, the second is how
//               much cash they currently have.
// Postcondition: A report detailing the number of chips
//                and the amount of cash the user has is
//                printed.
void statusreport(int numchips, int cash);
```

Restrictions

Name the file you create and turn in *casino.c*. Although you may use other compilers, your program must compile and run using gcc in Code::Blocks. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. You should also include comments throughout your code, when appropriate. If you have any questions about this, please see a TA.

Input Specification

Assume that the user always enters the proper ***type*** of input, but not always appropriate values. Here are some possible errors you need to check for:

1) Do not allow the user to spend more money on chips than they have.
2) Do not allow the user to bet a number of chips they don't have.
3) Do not allow the user to sell a number of chips they don't have.
4) The user must always bet at least one chip for a game, or they can not play the game.

Assume that these specifications will be followed by the user:

1) They will never enter any negative integers.
2) They will never enter any invalid menu choices.

Output Specification

Your output should follow the examples on the following pages.

Deliverables

A single source file named *casino.c* turned in through WebCourses. This file should be an edited version of casino_skeleton.c, which is posted with this assignment description.

Example Output #1

Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
1
How much cash do you want to spend for chips?
1100
Sorry, you do not have that much money. No chips bought.
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
1
How much cash do you want to spend for chips?
500
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
5
You currently have $505 left and 45 chips.
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
3
How many chips would you like to bet?
0
Sorry, that is not allowed. No game played.
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips

3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
3
How many chips would you like to bet?
10
Press 'r' and hit enter for your first roll.
r
You rolled a 8.
Press 'r' and hit enter for your next roll.
r
You rolled a 12.
Press 'r' and hit enter for your next roll.
r
You rolled a 6.
Press 'r' and hit enter for your next roll.
r
You rolled a 7.
Sorry, you have lost.
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
4
How many chips would you like to bet?
17
Press 'r' and hit enter for your first roll.
r
You rolled a 3.
Press 'r' and hit enter for your next roll.
r
You rolled a 8.
You win!
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
6
After selling your chips, you have $1025. Thanks for playing!

Example Output #2

Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
1
How much cash do you want to spend for chips?
500
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
2
How many chips do you want to sell?
46
Sorry, you do not have that many chips. No chips sold.
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
4
How many chips would you like to bet?
46
Sorry, you do not have that many chips to bet. No game played.
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
4
How many chips would you like to bet?
5
Press 'r' and hit enter for your first roll.
r
You rolled a 11.

You win!
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
4
How many chips would you like to bet?
10
Press 'r' and hit enter for your first roll.
r
You rolled a 5.
Press 'r' and hit enter for your first roll.
r
You rolled a 5.
Sorry, you have lost.
Welcome to the Casino. Here are your choices:
1) Buy chips
2) Sell chips
3) Play Craps
4) Play Arup's Game of Dice
5) Status Report
6) Quit
6
After selling your chips, you have $905. Thanks for playing!

COP 3223 Fall 2013 Grading Criteria Program 9
Total: 100 points

Execution Points (50 pts)

1) Buy Chips - 10 pts
2) Sell Chips - 10 pts
3) Craps - 25 pts
   - Win on 7 or 11 - 5 pts
   - Lose on 2, 3 or 12 - 5 pts
   - Continue Playing on other - 5 pts
   - Play continues until point or 7 – 5 pts
   - Correct winner is established - 5 pts
4) Status Report – 5 pt

Run each of these functions on the menu at various times, keeping track of what should occur for each. Only deduct a whole number of points. Use your judgement in awarding partial credit for each category. *Note: if a program doesn't compile, but all the logic is correct, you may award upto 25 of the 50 execution points. This will be left up to your mercy and discretion.*

Points for Code (30 pts)

1) pairofdice - 5 pts. Give them only 3 pt if the return a random value in between 2 and 12. Award the full 5 points if they produce two rand nums (1-6) and return their sum.

2) craps - 15 pts. 5 pts for tracking an immediate win or loss properly, 5 pts for properly assigning a loss if a 7 is hit first after a point, 5 pts for properly assigning a win if the point is rolled again.

3) buychips - 5 pts. 2 pt for determining the proper number of chips, 3 pt for adjusting the amount of cash given to equal the proper amount of change.

4) sellchips - 5 pts. 2 pts for using the number of chips passed in, 3 pts for returning their value.

Points for Comments & Style (20 pts)

1) Header comment with name, course number, section number, assignment title, and date. (2 pts)
2) Indentation (3 pts)
3) Use of white space (3 pts)
4) Appropriate variable names (3 pts)
5) Internal Comments (9 pts)

General note: Please record the score as an integer.
When you give the students comments, just tell them how much you took off and for what. Also, include your initials at the end of your comment so students know who graded their assignment.

Introduction to Programming (COP 3223) Assignment #7
File Input: Programming Contest problems.
Due date: *Please consult WebCourses*

Objectives
1. Learn how to read input from files and write output to files.
2. Review the use of if statements and loops **in C.**

Programming Contests: Background
In the course textbook, a brief history of programming contests is covered. UCF hosts a programming contest for high school students. In order to test students' programs on many possible input cases, we ask students to write programs that read in their data from a file (without prompting for the name of the file or for any of the information). Students must then read in the input from the file, calculate a solution to each case of input in the file and output the results in the exact format requested to the screen.

Your Assignment
For this assignment you will solve two of the easier problems we have posed in our high school contests. Since I want you getting practice writing to an output file as well, this assignment will consist of two programs. For the first program, Chalice, you will just write your output to the screen. **For your second program, Ice, please write your output to a file called out.txt. For both programs, please read your input from the files chalice.in and ice.in, respectively.**

Problem Descriptions
The descriptions of both problems are separate .pdf files: Chalice.pdf and Ice.pdf, exactly as they appeared in their original contest. Here is a summary of each section of the description:

The Problem – This states what problem you will be solving.
The Input – This states the exact format of the input file.
The Output – This states the exact format you must provide the output.
Sample Input – A sample input which is NOT comprehensive, but shows an example of a valid input file that adheres to the specification in "The Input"
Sample Output – The output your program should produce corresponding to the Sample Input.

Deliverables
Two source files:

1) *chalice.c*, for your solution to Problem A (Chalice.pdf),
   which reads from chalice.in and which writes to the screen.
2) *ice.c,* for your solution to Problem B (Ice.pdf),
   which reads from ince.in and which writes to the file out.txt.

All files are to be submitted over WebCourses.

Restrictions

Although you may use other compilers, your program must compile and run using Code::Blocks with gcc. Each of your three programs should include a header comment with the following information: your name, course number, section number, assignment title, and date. Also, make sure you include comments throughout your code describing the major steps in solving the problem.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness
2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.
3) Compatibility to Code::Blocks (in Windows). If your program does not compile in this environment, **the maximum credit you will receive is 50%.**

Final Note

I will be nice this time and provide our real testing data for the problem Ice, the harder of the two problems. My hope is that by providing you this data, you will expedite the debugging process by more quickly determining cases you hadn't thought of.

Montgomery Anaconda and the Sacred Chalice

Filename: `chalice`

By divine proclamation, Arturia was named ruler of all of Albion when she was given the holy saber, Excelsior, by the Woman in the Water. Unfortunately, the ignorant peasants, who consider themselves an autonomous autocracy, feel that her claim to rule is not proper for supreme executive power. Blasted peasants! In an attempt to prove herself as true ruler of the Brettons, the queen has undertaken a quest by the Big Guy in the Sky to search for the Sacred Chalice. Thus, Queen Arturia and her Sires of the Pentagonal Ottoman have begun their quest to find the Sacred Chalice, so that Arturia may claim her rightful place as Queen of the Brettons.

While on her quest, she comes across a village where a crowd of peasants are waving torches and pitchforks around a woman dressed in robes and a carrot on her nose, with loud chants of "She's a witch! Burn her!" being heard. Arturia's most intelligent sire, Bolivar, stepped forward to assess the situation. According to the villagers, this woman is a witch, and must, therefore, be burned. Unconvinced, Sire Bolivar suggested that there should be some test to ensure that she is a witch. But what kind of test should there be? Bolivar reasoned that if she is a witch, she is able to be burned. Because she can be burned, she must also be made of wood since it also burns. And if she is made of wood, she must also float. With that, the villagers began to ponder tossing her into the water to see if she'd float, only to realize that there is no large body of water nearby.

"Well", Bolivar then asked, "If she can float, she must be very light, at least as light as what?" The peasants stood dumbfounded. Then, the queen responded "A flock of geese!" to which Bolivar agreed whole-heartedly. Thus, if the woman truly is a witch, she must be as light as or lighter than a flock of geese. The crowd shouted in earnest and glee to test the woman of witchcraft. Unfortunately, their scales are not large enough to measure the woman and the flock of geese; thus, each must be weighed individually. Even worse, none of them are able to tell whether the woman or the flock of geese is heavier. Therefore, Queen Arturia has turned to you, Sire Not-Named-In-This-Problem, to use your programming skills to solve this dilemma, because here in 12th century Albion, no one else knows what in the world a computer is.

The Problem:

Given the weight in pounds of a person accused of witchcraft and the weight (also in pounds) of each of the geese in the trial's flock, determine whether or not the person is a witch, according to Sire Bolivar's reasoning. Since many other potential witches may be found, Queen Arturia would like you to make sure the program will work for multiple trials.

The Input:

The first line shall list a single positive integer, *n*, on a line by itself, stating the number of trials to be held. For each trial, the first line shall contain a single positive integer, *w* ($w \leq 500$), stating the weight of the accused. The next line shall contain a single integer, *f* ($1 \leq f \leq 10$), stating the number of geese in the flock. The next line shall contain *f* positive integers, each separated by a single space, telling the weight of each individual goose. No individual goose shall exceed 50 pounds.

The Output:

For each trial, output a line "`Trial #`*x*`: `*j*" where *x* is the current trial number (starting at 1), and *j* is either the phrase "`SHE'S A WITCH! BURN HER!`" if the accused is a witch according to her trial or "`She's not a witch. BURN HER ANYWAY!`" if she isn't. There should be one space after the colon.

Sample Input:

```
2
120
5
30 24 35 50 45
200
3
40 24 60
```

Sample Output:

```
Trial #1: SHE'S A WITCH! BURN HER!
Trial #2: She's not a witch. BURN HER ANYWAY!
```

Ali's Ice Cold Conundrum

Filename: `ice`

Ali is throwing a party at one of his houses, and he's laid out a big selection of drinks for everyone. It's pretty hot outside, so the ice maker is cranked up to the max. However, as Ali is dispensing ice, he notices that the ice maker tends to release ice cubes in small bursts of two to ten cubes each. Being the consummate host, Ali knows that seven cubes is the perfect number of cubes for the glasses he's using, but he's having trouble hitting that mark with the dispenser.

The Problem:

Given the number of ice cubes dispensed in each burst in order, tell Ali whether or not each cup has the perfect number (7 cubes). If not, let Ali know how many cubes he would have to remove to get the perfect number. Since his guests are thirsty, he won't waste time actually correcting the number of ice cubes in each cup (he just wants to keep track of how he's doing). Once each cup is full (7 or more cubes), Ali will automatically move on to the next cup. Ali will keep filling cups with ice as long as the ice maker dispenses cubes (once it runs out, Ali will wait for it to fill up again). Note that once the ice maker runs out of ice, the last cup might not have enough ice cubes in it. If so, let Ali know how many cubes that cup is missing. After each session, he will immediately serve all of the cups he has filled (whether they have 7 cubes or not).

The Input:

The ice maker will be filled and dispensed several times today. The input will begin with a single positive integer, s, on a line by itself, indicating the number of ice dispensing sessions. Each session will start with an integer, b ($1 \le b \le 50$), on a line by itself, indicating the number of ice dispensing bursts in this session. Following this, on each of the next b lines, there will be a single integer, c ($2 \le c \le 10$), indicating the number of ice cubes in this burst.

The Output:

At the start of each session, output a line of the form "`Session #`*n*`:`", where n is the number of the ice dispensing session (beginning at 1). For each filled cup in this session output a line indicating the state of that cup:

- If the cup has exactly 7 cubes, print "`Cup #`*i*`: Perfect!`"
- If the cup has too many cubes, print "`Cup #`*i*`:` *k* `cubes too many!`"
- If the cup has too few cubes, print "`Cup #`*i*`: Need` *k* `more cubes!`"

In all cases, i is the number of the filled cup in that session (beginning at 1), and k is the absolute value of the difference between the number of ice cubes dispensed into that cup and 7. The output line for each cup should be indented three spaces and there should be one space after the colon. Leave a blank line after the output for each dispensing session.

Sample Input:

```
2
3
10
2
5
4
2
2
3
2
```

Sample Output:

```
Session #1:
   Cup #1: 3 cubes too many!
   Cup #2: Perfect!

Session #2:
   Cup #1: Perfect!
   Cup #2: Need 5 more cubes!
```

COP 3223 Fall 2013 Grading Criteria Program 7
Total: 100 points

Problem A(50 pts)

Execution Points (20 pts)
1 point per test case (there are 20 in the file)

Note: THEY GET CREDIT HERE EVEN IF THEY DIDN'T FOLLOW THE EXACT OUTPUT SPELLING/FORMAT. THIS IS JUST FOR CORRECTNESS NOT FORMAT. THE FORMAT POINTS WILL BE IN THE NEXT SECTION.

Note: if a program doesn't compile, but all the logic is correct, you may award upto 4 of the 8 execution points. This will be left up to your mercy and discretion.

Also, if none of the test cases work because of a simple logical error, feel free to deduct 1 to 4 points for that logical error.

Points for Code (24 pts)
1) Input file is opened. (5 pts)
2) Input is properly read from file. (5 pts – can give partial)
3) Output is to standard out. (4 pts)
4) An accumulator variable exists. (5 pts)
5) Output format is followed. (5 pts – can give partial)

Points for Comments & Style (6 pts)
1) Header comment with name, course number, section number, assignment title, and date. (1 pts)
2) Indentation and use of white space (1 pt)
3) Appropriate variable names (2 pt)
4) Internal comments (2 pt)

Problem B (50 pts)

Execution Points (21 pts)
There are 42 sessions, ½ a pt per session, round down.

Note: if a program doesn't compile, but all the logic is correct, you may award upto 10 of the 20 execution points. This will be left up to your mercy and discretion. Also, if none of the test cases work because of a simple logical error, feel free to deduct 1 to 4 points for that logical error (based on your assessment of its severity) instead of the full 20 points for all the test cases.

Points for Code (23 pts)
1) Input file is opened. (4 pts)
2) Input is properly read from file. (4 pts – can give partial)
3) Output is to out.txt. (4 pts)
4) Output does NOT occur for every burst (3 pts)
5) Accumulator variable is used. (3 pts)
5) Output format is followed. (5 pts – can give partial)

Points for Comments & Style (6 pts)
1) Header comment with name, course number, section number, assignment title, and date. (1 pts)
2) Indentation and use of white space (1 pt)
3) Appropriate variable names (2 pt)
4) Internal comments (2 pt)

Introduction to Programming (COP 3223) Section 4 – Program #5

Due date: Please consult WebCourses for your section

Notes

1. Please read the notes on Code::Blocks.

Objectives

1. To give students practice at typing in, compiling and running simple programs.
2. To learn how to read in input from the user.
3. To learn how to use assignment statements and arithmetic expressions to make calculations
4. To learn how to use the if-then-else construct.

Problem A: Building a Goldfish Tank

You want to build a goldfish tank and determine how much it will cost. Each tank has three dimensions: length, width and height. The cost of building the tank is simply the cost of the five panes of glass that comprise the tank. (The top of the tank is not enclosed by glass.) Glass costs exactly $0.02 for one square inch. Define a constant to store this value in your program. Your program should prompt the user for the length, width and height of the tank in inches. Then your program should calculate the cost of building the tank and output that to the screen.

Input Specification

1. The length, width and height of the tank will be positive integers.

Output Specification

Output the cost in dollars to build the fish tank to two decimal places. Your output should follow the format below, where XX.XX is the cost in dollars to build the goldfish tank.

```
The goldfish tank costs $XX.XX to build.
```

Output Sample

Below is one sample output of running the program. Note that this sample is NOT a comprehensive test. You should test your program with different data than is shown here based on the specifications given above. In the sample run below, for clarity and ease of reading, the user input is given in *italics* while the program output is in bold.

Sample Run #1

What is the length of your goldfish tank in inches?
24
What is the width of your goldfish tank in inches?
12
What is the height of your goldfish tank in inches?
16
Your goldfish tank costs $28.80 to build.

Problem B: Maintaining the Tank

In order to actually use the goldfish tank you have built, you must maintain it. The cost of maintaining the tank is the volume of the tank multiplied by $0.005, (half of a penny). Write a program to read in the length, width and height of the tank and output the cost of maintaining the tank.

Input Specification

1. The length, width and height of the tank will be positive integers.

Output Specification

Output the cost in dollars to maintain the fishtank to two decimal places. Your output should follow the format below, where XX.XX is the cost in dollars to maintain the goldfish tank.

```
The goldfish tank costs $XX.XX to maintain.
```

Output Samples

Below are two sample outputs of running the program. Note that these samples are NOT a comprehensive test. You should test your program with different data than is shown here based on the specifications given above. In the sample run below, for clarity and ease of reading, the user input is given in *italics* while the program output is in bold.

Sample Run #1

What is the length of your goldfish tank in inches?
24
What is the width of your goldfish tank in inches?
12
What is the height of your goldfish tank in inches?
16
Your goldfish tank costs $23.04 to maintain.

Sample Run #2

What is the length of your goldfish tank in inches?
96
What is the width of your goldfish tank in inches?
12
What is the height of your goldfish tank in inches?
20
Your goldfish tank costs $115.20 to maintain.

Problem C: Determining the Profit of the Goldfish Tank

Using your solutions to problems A and B, you will solve the following problem:

Given the length, width and height of the goldfish tank in inches, determine the amount of profit you can gain by selling the fish in the tank. Assume that each goldfish in the tank requires 250 cubic inches of room in the tank to survive. (Thus, if the dimensions of the tank were 24x12x16 inches, exactly 18 fish could fit in the tank since 24x12x16 = 4608 and 4608/250 = 18.432, but you can't have .432 of a fish.) Also, assume that you sell each goldfish for $5.00. (You can store these values in constants in your program.)

Here's an example worked out:

If the dimensions of the tank are 24x12x16, we have already determined that the cost of building the tank is $28.80 and the cost of maintaining the tank is $23.04, so the total cost is $51.84.

But, we will sell 18 goldfish at $5.00 dollars a piece for a total gross revenue of $90.00. Thus, our profit is $90.00 - $51.84 = $38.16.

There is also a possibility that you may lose money with your fishtank. If this occurs, print out a message (following the sample below) indicating the amount of money lost.

Input Specification

1. The length, width and height of the tank will be positive integers.

Output Specification

Output the profit in dollars for selling goldfish to two decimal places. If there was a profit, your output should follow the format below, where XX.XX is the profit from selling the goldfish.

```
Your profit from selling goldfish is $XX.XX.
```

Otherwise, if there was a loss, you should follow the following format:

```
Your loss from selling goldfish is $XX.XX.
```

You may choose to handle the case where there was no profit or loss in any way you see fit.

Output Samples

On the next page are two sample outputs of running the program. Note that these samples are NOT a comprehensive test. You should test your program with different data than is shown here based on the specifications given above. In the sample run below, for clarity and ease of reading, the user input is given in *italics* while the program output is in bold.

Sample Run #1

```
What is the length of your goldfish tank in inches?
24
What is the width of your goldfish tank in inches?
12
What is the height of your goldfish tank in inches?
16
Your profit from selling goldfish is $38.16.
```

Sample Run #2

```
What is the length of your goldfish tank in inches?
1
What is the width of your goldfish tank in inches?
1
What is the height of your goldfish tank in inches?
1000
Your loss from selling goldfish is $65.02.
```

Deliverables

Three source files:

1) *tank.c*, for your solution to problem A
2) *maintain.c* for your solution to problem B
3) *profit.c* for your solution to problem C

All files are to be submitted over WebCourses.

Restrictions

Although you may use other compilers, your program must compile and run using gcc in Code::Blocks. Each of your three programs should include a header comment with the following information: your name, course number, section number, assignment title, and date. Also, make sure you include comments throughout your code describing the major steps in solving the problem.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness
2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.
3) Compatibility to Code::Blocks (using gcc). If your program does not compile in either of these environments, you will get a **sizable** deduction from your grade.

COP 3223 Fall 2013 Grading Criteria Program 5
Total: 100 points

Problem A(35 pts)
Execution Points (20 pts)
1) Prompts user for appropriate input values (2 pts)
2) Properly reads in the input values (4 pts)
3) Properly formats the output (2 pts)
3) Properly executes on each of the test cases below (4 pts for each case)

a) length = 48, width = 24, height = 18 (74.88)
b) length = 15, width = 10, height = 12 (15.00)
c) length = 100, width = 20, height = 50 (280.00)

Note: if a program doesn't compile, but all the logic is correct, you may award upto 6 of the 12 execution points. This will be left up to your mercy and discretion.

Also, if none of the test cases work because of a simple logical error, feel free to deduct 1 to 4 points for that logical error (based on your assessment of its severity) instead of the full 12 points for all the test cases.

Points for Code (11 pts)
1) Properly included #defines (2 pt)
2) Calculated the area of FIVE sides of the tank (5 pts)
3) Multiplied by the per square inch (2 pt)
3) printf's and scanf's were located in the proper places (2 pt)

Points for Comments & Style (4 pts)
1) Header comment with name, course number, section number, assignment title, and date. (1 pts)
2) Indentation and use of white space (1 pt)
3) Appropriate variable names (1 pt)
4) Internal comments (1 pt)

Problem B (30 pts)
Execution Points (15 pts)
1) Prompts user for appropriate values and reads them in correctly (3 pts)
2) Test cases: 4 pts for each of these cases:

a) length = 48, width = 24, height = 18 (103.68)
b) length = 15, width = 10, height = 12 (9.00)
c) length = 100, width = 20, height = 50 (500.00)

Note: if a program doesn't compile, but all the logic is correct, you may award upto 6 of the 12 execution points. This will be left up to your mercy and discretion.

Also, if none of the test cases work because of a simple logical error, feel free to deduct 1 to 4 points for that logical error (based on your assessment of its severity) instead of the full 12 points for all the test cases.

Points for Code (11 pts)
1) Properly included #defines (2 pt)
2) Used logically sound formula to find the volume (5 pt)
3) Multiplied by cost per cubic inch of the tank (2 pt)
4) printf's and scanf's were located in the proper places (2 pt)

Points for Comments & Style (4 pts)
1) Header comment with name, course number, section number, assignment title, and date. (1 pts)
2) Indentation and use of white space (1 pt)
3) Appropriate variable names (1 pt)
4) Internal comments (1 pt)

Problem C(35 pts)
Execution Points (20 pts)
1) Prompts user for appropriate input values (2 pts)
2) Properly reads in the input values (4 pts)
3) Properly formats the output (2 pts)
3) Properly executes on each of the test cases below (4 pts for each case)
    a) length = 48, width = 24, height = 18 (231.44)
    b) length = 5, width = 10, height = 1000 (149.00)
    c) length = 2, width = 2, height = 500 (50.08)

Note: if a program doesn't compile, but all the logic is correct, you may award upto 4 of the 12 execution points. This will be left up to your mercy and discretion.

Also, if none of the test cases work because of a simple logical error, feel free to deduct 1 to 4 points for that logical error (based on your assessment of its severity) instead of the full 12 points for all the test cases.

Points for Code (11 pts)
1) Properly included #defines (2 pt)
2) Used a mathematically sound formula for the number of fish in the tank (5 pts) (Give them points here even if they use integer division.)
3) Multiplied number of fish by $5.00 (2 pt)
3) printf's and scanf's were located in the proper places (2 pt)

Points for Comments & Style (4 pts)
1) Header comment with name, course number, section number, assignment title, and date. (1 pts)
2) Indentation and use of white space
3) Appropriate variable names (1 pt)

4) Comments in code (1 pt)

General note: Please record the score as an integer. To ensure this, never take off fractional points, either take a point off or don't. A good way to balance this out if you feel like taking off fractional points is if you see a two mistakes worth .5 points each, just take off 1 point for 1 of them.

When you give the students comments, just tell them how much you took off and for what. Also, include your initials at the end of your comment so students know who graded their assignment.

COP 3223 Program #2:

Due date: Please consult WebCourses for your section

Objectives

1. To practice the if statement (in python).
2. To learn how to look up math function calls and use these in a program.
3. To learn how to long up random function calls and use these in a program.
4. To learn how to use a basic for loop and use it in a program.

Introduction: Big Money

Many students like playing the lottery, since it's one of the things they are allowed to do once they turn 18. However, probabilistically, lotteries are not good investments. In fact, many mathematicians often refer to them as a "stupid tax". In this assignment you will take a look at figures simulating the Florida lottery and make up your own mind!

Part A: Calculate Winnings (winnings.py)

In the regular Florida Lotto, players select 6 unique integers in the range [1, 52] for each ticket they buy. The winnings for a ticket are based on how many matching numbers are on a ticket. Players win money if they match 3, 4, 5 or all 6 numbers. In this program, you will ask the user how many numbers they matched and output their winnings. For the purposes of this assignment, please use the actual winnings for the August 31, 2013 Lotto, which are as follows:

3 numbers: $5.00
4 numbers: $49.50
5 numbers: $4189.00
6 numbers: $3,000,000.00

Input Specification

The one input value will be a positive integer in between 0 and 6, inclusive, representing the number of matched numbers on a single ticket.

Output Specification

Output a single line with the following format:

```
You win $X.
```

where X represents the winnings for the given ticket.

Sample Program Run (User Input in Bold)

```
How many numbers did you match on your ticket?
```
4
```
You win $49.50.
```

Part B: Probability of Winning (prob.py)

For this portion of the assignment you will write a program that calculates the probability of matching a certain number of values on a single lottery ticket. In particular, we can use some basic probability to show that if we are choosing from the set of integers in the range [1, n] and there is a single combination of k integers from the set that we want to match, our probability of correctly choosing exactly m of those k integers in a single combination of k integers is:

$$\frac{\binom{k}{m}\binom{n-k}{k-m}}{\binom{n}{k}}$$

Input Specification

The one input value will be a positive integer in between 0 and 6, inclusive, representing the number of matched numbers on a single ticket.

Output Specification

Output a single line with the following format:

```
The probability of matching X values is Y.
```

where X is the user's input value and Y is the corresponding probability as a decimal. (Thus, if the probability was 1%, you should print 0.01 or an equivalent for the result.)

Sample Program Run (User Input in Bold)

```
How many numbers do you want to match on your ticket?
4
Your probability of matching 4 values is 0.0007625799910799017.
```

Part C: Lottery Simulation (sim.py)

In this part of the program you will use some given code to simulate buying lots of lottery tickets to see what your winnings would be. Since you don't know how to use lists yet, which are very helpful for the simulation, you will be given code to use for your simulation. All you have to do is use the given functions and simulate buying the desired number of tickets to determine the return on investment. The given code will be in a separate posted file, simhelp.py. Please copy and paste this code into your solution, sim.py.

In running your simulation, please use the payoffs from part A.

Input Specification

The one input value will be a positive integer, representing the number of tickets the user wants to buy. (Each ticket costs $1.)

Output Specification

If the user made money off buying the tickets, output a single line with this format:

```
You won $X, for a net earnings of $Y.
```

Otherwise, if the user lost money off buying the tickets, output a single line with this format:

```
You won $X, for a net loss of $Y.
```

Sample Program Run (User Input in Bold)

```
How many tickets do you want to buy?
1000000
You won $187612.0 for a net loss of $812388.0
```

Note: it takes a few seconds for python to generate a million lottery tickets and check for matches using the given code (which is somewhat inefficient).

Deliverables

Three source files: *winnings.py*, for your solution to problem A, *prob.py* for your solution to problem B, and *sim.py* for your solution to problem C. All files are to be submitted over WebCourses.

Restrictions

Although you may use other compilers and coding environments, your program must run in IDLE.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

Fall 2013 Section 4 COP 3223 Program #1 Grading Criteria

Part A: Winnings (30 pts)

Coding Style Points (6 pts)
Written in a separate file - 1 pt
Uses main function - 1 pt
Header comment - 1 pt
Internal comments - 1 pt
Declare constants (either beginning of main or before main) - 1 pt
Good variable names - 1 pt

Code Points (12 pts)
Reads in integer input (4 pts)
Has if statement (4 pts)
Prints Output (4 pts)

Test Cases (10 pts)
2 pts per case: Try inputs 1, 3, 4, 5 and 6. The answers are 0, 5, 49.50, 4189, 3000000. Note that it's not necessary to output to two decimal places. If they say "You don't win anything" for input 1, take off the 2 points since they didn't follow the spec.

Part B: Probability of Winning (40 pts)

Coding Style Points (6 pts)
Written in a separate file - 1 pt
Uses main function - 1 pt
Header comment - 1 pt
Internal comments - 1 pt
Declare constants (either beginning of main or before main) - 1 pt
Good variable names - 1 pt

Code Points (13 pts)
Reads in integer input (3 pts)
Imports Math (2 pts)
Calls factorial (2 pts)
Has a formula for a combination somewhere (4 pts)
Outputs a result (2 pts)

Test Cases (21 pts)
3 pts per test case – try each input. Match answers to solution.

Part C: Lottery Simulation (30 pts)

Coding Style Points (6 pts)
Written in a separate file - 1 pt
Uses main function - 1 pt
Header comment - 1 pt
Internal comments - 1 pt
Declare constants (either beginning of main or before main) - 1 pt
Good variable names - 1 pt

Code Points (14 pts)
Leaves my two functions untouched and includes them (4 pts)
Has a loop in main (4 pts)
Has an if statement inside the loop (2 pts)
Has an accumulator variable (4 pts)

Test Cases (10 pts)
Correctly generates multiple tickets and stores how many matches there were (4 pts)
Correctly accumulates winnings (3 pts)
Correctly outputs result based on whether you're up or down (3 pts)

Note – you have to look at the code to see if these things are done correctly. Run the program by buying a million tickets. Usually, you should lose about 800,000 but there can be significant variation…

COP 3223 Program #10: Minesweeper Setup (minesweeper.c)

Objective

To give students practice in writing functions and calling those functions to perform more complex tasks.

The Problem: Minesweeper Set Up

For this week you will only write one program. **Your program will read its input from the file, "mine.txt" and write output to the screen.**

This file will have information about multiple minesweeper boards. In particular, it will tell you the location of each mine on the board for your program to evaluate. Your program must read in this information and then print out a version of the board that has a '*' where each bomb is located and a number indicating the number of adjacent bombs for each non-bomb location.

The format of the input file is as follows:

The first line of the input file contains a single positive integer, *n*, representing the number of minesweeper boards to construct.

The first line of each board contains a single positive integer, *m*, representing the number of bombs for that board. The following *m* lines will contain two non-negative integers in between 0 and 7, inclusive, representing the row and column, respectively, of the location of a bomb. No two of these *m* lines will be identical.

For each board, output to the screen, the following header line:

```
Board #k:
```

where k is the number of the board, starting with 1.

Follow this with the board printed out over 8 lines, with a space in between each entry.

Separate the output for each case with a blank line.

Hints

Store the board in an 8 x 8 integer array. Store the number 9 where bombs are supposed to be located. When printing the board, use an if statement to screen for bombs. Both 8 and 9 should be represented by symbolic constants in this program. Watch out for array out of bounds issues by using if statements to screen for illegal locations. One neat idea to fill the board would be to initialize all non-bomb squares to 0 and then add one to each square adjacent to each bomb (except for other bombs!) The other alternative idea is to go to each non-bomb square and look at each location adjacent to it. Also, an inbounds function that takes in an x and y coordinate and returns whether or not it's inbounds may be helpful.

Sample Input

```
2
3
0 0
2 1
7 7
10
0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
7 0
7 1
```

Corresponding Sample Output

```
Board #1:
* 1 0 0 0 0 0 0
2 2 1 0 0 0 0 0
1 * 1 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 1 *

Board #2:
* * * * * * * *
2 3 3 3 3 3 3 2
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
2 2 1 0 0 0 0 0
* * 1 0 0 0 0 0
```

Restrictions

Although you may use other compilers, your program must compile and run using gcc in Code::Blocks. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. You should also include comments throughout your code, when appropriate. If you have any questions about this, please see a TA. **Your program must read its input from mine.txt.**

Deliverables

A single source file named *minesweeper.c* turned in through WebCourses.

Fall 2013 Section 4 COP 3223 Program #10 Grading Criteria

Minesweeper(100 pts total)

Coding Style Points (16 pts)

Header Comment – 2 pt
Internal Comments – 4 pts
Good variable names – 2 pt
Indented properly – 2 pt
Use of white space – 2 pt
Consistency of style – 2 pt
Uses constants for SIZE and BOMB – 2 pt

Code Points (24 pts)

Opens and Closes Input File – 1 pt
Loops for number of cases – 1 pt
Creates a board of the right size – 1 pt
Reads in correct number of bombs – 1 pt
Has code to avoid array out of bounds – 4 pts
Has code to print the board –43 pts
Has code to try to count bombs somehow – 4 pts
Has a reasonable function breakdown – 8 pts (you decide partial)

Execution Points(60 pts)

Compiles – 10 pts
There are 100 test cases, award 1 point for 2 correct test cases, for a maximum of 50 points, rounding down.

COP 3223 Program #11: Zip String (zip.c)

Objective

To give students practice manipulating strings in C.

The Problem: Zip String

For this week you will only write one program. **Your program will read its input from the file, "zip.txt" and write output to the screen.**

Zip String Algorithm for this assignment

There are many ways to compress data. One common general technique is run-length encoding. The basic idea here is when any character is repeated contiguously many times, instead of storing each copy of the character, simply store how many repetitions existed and one copy of the character.

For example, the string

AAAAAAAGCCCCCCCAAAAATTAATTAAC

would be compressed as:

7A1G7C5A2T2A2T2A1C

Input File Format

The first line of the file will contain a single positive integer, *n*, representing the number of strings to zip. The next *n* lines will each have one string of capital letters in between 1 and 29,999 characters long.

Output Format

For each input string, simply output the corresponding zipped string on a line by itself with no extra information.

For each board, output to the screen, the following header line:

```
Board #k:
```

Sample Input

```
3
AAAAAAAGCCCCCCCAAAAATTAATTAAC
B
ZZZZZZZZZZZZZZZZZZZ
```

Sample Output (Corresponding to Sample Input)

```
7A1G7C5A2T2A2T2A1C
1B
19Z
```

Implementation Requirements (for full credit)

1) Read each string into a character array of size 30000. Do NOT use variable length arrays.
2) You must write at least one function other than main.
3) Code must be follow good style and be well-commented.
4) No array out of bounds may occur.
5) Output must be correct.

Restrictions

Although you may use other compilers, your program must compile and run using gcc in Code::Blocks. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. You should also include comments throughout your code, when appropriate. If you have any questions about this, please see a TA. **Your program must read its input from zip.txt.**

Deliverables

A single source file named *zip.c* turned in through WebCourses.

Fall 2013 Section 4 COP 3223 Program #11 Grading Criteria

Zip (100 pts total)

Coding Style Points (15 pts)
Header Comment – 2 pt
Internal Comments – 4 pts
Good variable names – 2 pt
Indented properly – 2 pt
Use of white space – 2 pt
Consistency of style – 2 pt
Uses a constant for max length – 1 pt

Code Points (25 pts)
Opens and Closes Input File – 4 pt
Loops for number of cases – 3 pt
Creates a string that's big enough – 3 pt
Compares consecutive character somewhere – 5 pts
Has code to avoid array out of bounds – 5 pts
Has a reasonable function breakdown – 5 pts (you decide partial)

Execution Points(60 pts)
Compiles and doesn't crash – 10 pts
Has valid output for each case (string of number letter combos) – 10 pts
There are 20 test cases, award 2 points per case.

COP 3223 Program #3: Counting Pez

Part A: Maximum Possible Pez (maxpez.py)

Arup is embarking on an ambitious project: to fill ALL of his Pez dispensers. Normally, this would amount to just counting the number of dispensers and multiplying by 12, the maximum possible candies per dispenser. Unfortunately, Arup has collected some unique dispensers, some of which have different holding capacities of Pez! Write a program to quantify Arup's Pez needs.

When your program begins, prompt the user to enter the number of types of Pez dispensers he/she has. You are guaranteed that this will be a positive integer less than 10. Then, for each type of dispenser, ask the following questions:

1) How many Pez fit in one dispenser of this type?
2) How many dispensers of this type do you own?

You are guaranteed that the answers to both of these questions will be positive integers less than 1000. At the end of the program, print out the total number of Pez needed to fill the dispensers.

Sample Program Run #1

```
How many types of Pez do you have?
2
How many Pez fit in dispenser type #1?
10
How many dispensers of type #1 do you have?
4
How many Pez fit in dispenser type #2?
12
How many dispensers of type #2 do you have?
5
You need 100 Pez candies.
```

Sample Program Run #2

```
How many types of Pez do you have?
3
How many Pez fit in dispenser type #1?
1
How many dispensers of type #1 do you have?
4
How many Pez fit in dispenser type #2?
100
How many dispensers of type #2 do you have?
6
How many Pez fit in dispenser type #3?
20
How many dispensers of type #3 do you have?
4
You need 684 Pez candies.
```

Part B: Multiplying Pez (multpez.py)

It turns out that at the Pez manufacturing plant in Orange, Connecticut, the company has produced a super-secret type of Pez that procreates! Unfortunately, these new products have yet to be FDA approved, which is why you haven't seen them in stores yet. Because you are so excited about the new upcoming product, you decide that you want to write a program that can accurately calculate how many of these procreating Pez you'll have after a few days of buying an original collection.

In particular, each type of these Pez is given a "reproduction factor", R, which is a positive integer less than 10, as well as a period length, D, which is a positive integer number of days. Each of the current Pez pairs up with another Pez and produces R new Pez every D days. Of course, with all of these Pez running around, you must eat some of them. Let E be the number of Pez you eat every D days. Given these three factors, R, D and E, you can produce a chart showing the number of Pez around after D days, 2D days, and so forth.

Consider the example of Pez with a reproduction factor of 3, with a period length of 10 days, where you eat 5 Pez each 10 days and start with 20 Pez. This chart shows how many Pez are available on days 0, 10, 20 and 30

Day	Number of Pez
0	20
10	45
20	106
30	260

In between day 0 and day 10, there are 10 pair of Pez which produce 30 new Pez, for a total of 50. Of these, five are eaten by day 10 for a total of 45 remaining. From day 10 to 20, 22 pairs of Pez (one Pez is alone unfortunately :( ) produce 66 new Pez for a total of 111, of which 5 are eaten for a grand total of 106 at the end of day 20. Finally, 53 pairs of Pez in between day 20 and 30 produce 159 new Pez, of which 5 are eaten for a total of 260 Pez left (106 + 159 - 5) at the end of day 30.

For your program, prompt the user to enter the initial number of Pez (a positive integer less than 1000), the reproductive factor R (a positive integer less than 10), the period length D (a positive integer less than 100), the number of Pez eaten every D days, and an integer N (a positive integer less than 50), the number of periods for which we want the chart printed, assuming that the first line of the chart represents period 0.

Sample Program Run #1

```
How many reproducing Pez are you starting with?
20
What is their reproductive factor?
3
How long is a single reproductive cycle in days?
10
How many Pez do you eat every 10 days?
5
```

```
How many periods do you want the chart printed?
3

Day             Number of Pez
---             -------------
0               20
10              45
20              106
30              260
```

Note: You don't need to match the spacing of this chart exactly, but try to separate out the two columns with a couple tabs ('\t').

Part C: How Old are the Pez? (oldpez.py)

Not only can Pez dispense candy, but they can talk! One of them has given you a riddle that is too difficult for you to solve on your own, but you realize you can use your Python programming skills to write a program that will discover a solution. The riddle is of the following format: There are three Pez, Annie, Bobby and Carmen. You are given the sum of their ages and the product of their ages, and your goal is to figure out all possibilities for how old each of them is. Luckily, you know that all Pez surviving today are in between 1 and 86, inclusive! (Pez were first created by Eduard Haas III in 1927.)

Prompt the user to enter both the sum and product of the ages of the three Pez. Both values will be positive integers such that all possible solutions for the ages are in between 1 and 86, inclusive.

For each possible age setting, print out a single line with the following format:

```
Annie = A Bobby = B Carmen = C
```

where A, B and C are ages respectively, that satisfy the input values given.

Print your solutions in order, with A smallest to A largest, breaking ties by order of B, and breaking those ties by order of C.

Sample Program Run #1

```
What is the sum of Annie, Bobby and Carmen's ages?
20
What is the product of Annie, Bobby and Carmen's ages?
280
Annie = 5 Bobby = 7 Carmen = 8
Annie = 5 Bobby = 8 Carmen = 7
Annie = 7 Bobby = 5 Carmen = 8
Annie = 7 Bobby = 8 Carmen = 5
Annie = 8 Bobby = 5 Carmen = 7
Annie = 8 Bobby = 7 Carmen = 5
```

Deliverables

Three source files: *maxpez.py*, for your solution to problem A, *multpez.py* for your solution to problem B, and *oldpez.py* for your solution to problem C. All files are to be submitted over WebCourses.

Restrictions

Although you may use other compilers and coding environments, your program must run in IDLE.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

Fall 2013 Section 4 COP 3223 Program #3 Grading Criteria

Part A: Max Pez (30 pts)

Coding Style Points (5 pts)
Written in a separate file - 1 pt
Uses main function - 1 pt
Header comment - 1 pt
Internal comments - 1 pt
Good variable names - 1 pt

Code Points (10 pts)
Reads in integer inputs at appropriate times (4 pts)
Has loop (3 pts)
Prints Output (3 pts)

Test Cases (15 pts)
Three test cases, 5 pts each

1) 1, (5, 5) (Answer = 25)
2) 3, (2, 7), (10, 3), (12, 6) (Answer = 116)
3) 6 (8, 1), (1, 5), (16, 17), (100, 100), (18, 5), (3, 4) (Answer = 10297)

Part B: Multiplying Pez(35 pts)

Coding Style Points (5 pts)
Written in a separate file - 1 pt
Uses main function - 1 pt
Header comment - 1 pt
Internal comments - 1 pt
Good variable names - 1 pt

Code Points (10 pts)
Reads in all input (5 pts)
Has loop (3 pts)
Has print in loop (2 pts)

Test Cases (20 pts)
4 cases, 5 pts each – generate the charts with the solution…

1) 20, 3, 10, 5, 3 (chart in sample)
2) 10, 5, 8, 20, 10
3) 20, 2, 18, 20, 15
4) 1000, 1, 2, 502, 10

Part C: Old Pez (35 pts)

Coding Style Points (6 pts)
Written in a separate file - 1 pt
Uses main function - 1 pt
Header comment - 1 pt
Internal comments - 1 pt
Declare constant for max age- 1 pt
Good variable names - 1 pt

Code Points (14 pts)
Reads input (4 pts)
Has a double loop structure (or triple) (4 pts)
Checks for sum somehow (2 pts)
Checks for product somehow (2 pts)
Has print in loop (2 pts)

Test Cases (15 pts)
5 pts each

1) 13, 72 (6 answers, perms of 3, 4, 6)
2) 47, 2800 (9 solutions, two diff combos, 7, 20, 20 and 8, 14, 25)
3) 255, 614040 (6 answers, perms of 84, 85, 86)

Introduction to Programming (COP 3223) Assignment #6
Road Trip Planning
Due date: *Please consult WebCourses*

Objectives
1. Learn how to write and use loops **in C.**
2. Review the use of if statements **in C.**

Problem A: Rest Stops (rest.c)
When traveling by car, families usually stop for two reasons: (a) to get gas, (b) for a food and bathroom break. In this program, given how often a family must stop for each of these things, print out a list of the miles traveled at each stop before the family reaches its destination as well as the reason for the stop. For example, if we must stop every 30 miles to get gas and every 40 miles to get food on a 150 mile trip, we would stop at mile 30 for gas, mile 40 for food, mile 60 for gas, mile 80 for food, mile 90 for gas, and mile 120 for both.

Input Specification
The length of the road trip (in miles) will be a positive integer less than or equal to 5000. The distance between stops for gas and food will be positive integers less than or equal to 1000. (**Note: Please DO NOT have if statements to check to see if the entered values match these specs, simply assume that they do.)**

Output Specification
For each stop print out a line with one of the three following formats:

```
STOP AT MILE X FOR GAS
STOP AT MILE X FOR FOOD
STOP AT MILE X FOR BOTH
```

These should be printed out in the order of the stops.

Sample Run
```
How long is your road trip, in miles?
150
What is the distance between stops for gas, in miles?
30
What is the distance between stops for food, in miles?
40

STOP AT MILE 30 FOR GAS
STOP AT MILE 40 FOR FOOD
STOP AT MILE 60 FOR GAS
STOP AT MILE 80 FOR FOOD
STOP AT MILE 90 FOR GAS
STOP AT MILE 120 FOR BOTH
```

Problem B: Candy!!! (candy.c)

One nice thing about road trips is that your parents let you eat candy on the trip! You and your siblings have devised a fun game to determine how to split the candy. All of the candy starts in a pile. Then you alternate turns taking in between 1 and k pieces of candy, where k is a positive integer both of you have agreed upon. The person who takes the last piece "wins" and gets 75% of the candy! Write a program to simulate this game.

Input Specification

The number of pieces of candy to start the game will be a positive integer less than or equal to 100. The maximum number of pieces that can be taken per turn will be a positive integer less than or equal to 10. Assume both players will always take a valid number of pieces of candy. (**Note: Please DO NOT have if statements to check to see if the entered values match these specs, simply assume that they do.)**

Output Specification

After each turn, print out how many pieces of candy are left. Follow the sample output provided below.

Sample Run

```
How many pieces of candy are you starting with?
20
What is the maximum number of pieces per turn?
7
Player #1, how many pieces will you take?
4
There are 16 pieces left.
Player #2, how many pieces will you take?
5
There are 11 pieces left.
Player #1, how many pieces will you take?
3
There are 8 pieces left.
Player #2, how many pieces will you take?
1
There are 7 pieces left.
Player #1, how many pieces will you take?
7
There are 0 pieces left.
Player #1 wins!
```

Problem C: Game for the Kids (multgame.c)

Your car's computer is doing a great job keeping track of gas mileage and even has a neat feature that prints out a wheel, but it would be nice if the kids could use it in a constructive fashion. Write a program that drills the kids with multiplication problems. In particular, ask the user how many problems for the game, and then give the user that many random multiplication problems, where each number being multiplied is in between 0 and 12, inclusive. While the user is doing the problems, if they answer incorrectly, tell them so, and allow them to answer again. Do not move onto the next problem until they have correctly answered the current one. At the end of the game, when the user has correctly solved all the given multiplication problems, output the amount of time they spent.

How to calculate time spent for a segment of code in a C program:

In order to calculate how much time something takes, you can use the time function. In particular, the function call time(0) returns an int that represents the number of seconds after the birth of the Unix operating system. In order to effectively use this, you must call the function twice: once right before you start what you want to time, and once right afterwards. Subtract these two values to obtain the amount of time a segment of code took. Here is a short example:

```
int start = time(0);
// Insert code you want to time here.
int end = time(0);
int timespent = end - start;
printf("Your code took %d seconds.\n", timespent);
```

Input Specification

The number of problems to answer entered by the user will always be a positive integer less than 50.

Output Specification

For each correct response, simply move onto the next problem. For each incorrect response, output a message with the following format.

```
Incorrect, try again. AxB =
```

where A and B are the two numbers to multiply from the original problem.

After all the problems are completed, output a single line with the following format:

```
You completed X problems in Y seconds.
```

where X is the number of problems solved and Y is the number of seconds it took the user to solve them.

Output Samples

Here is one sample output of running the program. Note that this sample is NOT a comprehensive test. You should test your program with different data than is shown here based on the specifications given above. The user input is given in *italics* while the program output is in bold.

Sample Run #1

```
How many problems do you want?
5
Answer: 3x9 = 27

Answer: 4x6 = 42
Incorrect, try again.
Answer: 4x6 = 24

Answer: 12x11 = 132

Answer: 8x2 = 16

Answer: 7x5 = 35

You completed 5 problems in 17 seconds.
```

Deliverables

Three source files:

1) *rest.c*, for your solution to Problem A
2) *candy.c*, for your solution to Problem B
3) *multgame.c*, for your solution to Problem C

All files are to be submitted over WebCourses.

Restrictions

Although you may use other compilers, your program must compile and run using Code::Blocks with gcc. Each of your three programs should include a header comment with the following information: your name, course number, section number, assignment title, and date. Also, make sure you include comments throughout your code describing the major steps in solving the problem.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness
2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.
3) Compatibility to Code::Blocks (in Windows). If your program does not compile in this environment, **the maximum credit you will receive is 50%.**

COP 3223 Fall 2013 Grading Criteria Program 6
Total: 100 points

Problem A(30 pts)

Execution Points (18 pts)
3 points for each of the following six test cases:

1) 150, 30, 40 (see sample)
2) 400, 40, 80 (both every other)
3) 400, 80, 40 (bothe very other)
4) 20, 1, 1 (both on all)
5) 27, 5, 7 (one or the other)
6) 1000, 78, 130

Note: if a program doesn't compile, but all the logic is correct, you may award upto 4 of the 8 execution points. This will be left up to your mercy and discretion.

Also, if none of the test cases work because of a simple logical error, feel free to deduct 1 to 4 points for that logical error.

Points for Code (8 pts)
1) All input values are read in (2 pts)
2) A loop is in the code (2 pts)
3) All three possible prints are somewhere in the code (2 pts)
4) If statements appear somewhere in the code (2 pts)

Points for Comments & Style (4 pts)
1) Header comment with name, course number, section number, assignment title, and date. (1 pts)
2) Indentation and use of white space (1 pt)
3) Appropriate variable names (1 pt)
4) Internal comments (1 pt)

Problem B (35 pts)
Execution Points (20 pts)
1) Prompts user for appropriate values and reads them in correctly (4 pts)
2) Test cases: 4 pts for each of these cases:

a) 20, 4 (parameters) game: 4, 4, 4, 4, 4
b) 5, 5 (parameters) game: 5
c) 6, 5 (parameters) game: 5, 1
d) 15, 3 (parameters) game: 2, 3, 2, 3, 2, 3

Take off 2 points total if the only incorrect thing is the # of the player who won.

Note: if a program doesn't compile, but all the logic is correct, you may award upto 10 of the 20 execution points. This will be left up to your mercy and discretion. Also, if none of the test cases work because of a simple logical error, feel free to deduct 1 to 4 points for that logical error (based on your assessment of its severity) instead of the full 20 points for all the test cases.

Points for Code (11 pts)

1) There is a loop. (3 pts)
2) Amount of candy is adjusted. (3 pts)
3) Method to exit the loop depends on candy left. (3 pts)
4) There is a print for the winner. (2 pts)

Points for Comments & Style (4 pts)

1) Header comment with name, course number, section number, assignment title, and date. (1 pts)
2) Indentation and use of white space (1 pt)
3) Appropriate variable names (1 pt)
4) Internal comments (1 pt)

Problem C(35 pts)

Execution Points (22 pts)

1) Prompts user for appropriate input values and reads them in (2 pts)
2) Properly does the following while executing:
    a) Gives multiplications of numbers from 0 to 12. (4 pts)
    b) Gives the correct number of multiplications. (4 pts)
    c) Properly checks if your answer is correct. (4 pts)
    d) Prompts with same problem if incorrect answer. (4 pts)
    e) Prints out the time it took to complete the problems. (4 pts)

Note: if a program doesn't compile, but all the logic is correct, you may award upto 11 of the 22 execution points. This will be left up to your mercy and discretion.

Also, if none of the test cases work because of a simple logical error, feel free to deduct 1 to 4 points for that logical error (based on your assessment of its severity) instead of the full 22 points for all the test cases.

Points for Code (9 pts)

1) Uses loop to iterate through all problems (3 pt)
2) Uses loop for each question until correct (3 pt)
3) Has a mechanism for keeping track of time (3 pts)

Points for Comments & Style (4 pts)

1) Header comment with name, course number, section number, assignment title, and date. (1 pts)
2) Indentation and use of white space
3) Appropriate variable names (1 pt)
4) Comments in code (1 pt)

General note: Please record the score as an integer. To ensure this, never take off fractional points, either take a point off or don't. A good way to balance this out if you feel like taking off fractional points is if you see a two mistakes worth .5 points each, just take off 1 point for 1 of them.

When you give the students comments, just tell them how much you took off and for what. Also, include your initials at the end of your comment so students know who graded their assignment.

COP 3223 Program #12: Stock Sorting

Objective

To give students practice designing and using their own struct to solve a problem.

The Problem: Sorting Stocks

In this program you will read in stock data for the S&P 500 Stocks from the input file "stockdata.txt" and you'll ask the user how they would like the data sorted, providing the following options:

1) By value of change from the first day of the interval to the last, with largest positive change coming first.

2) By average value over the interval (from highest to lowest)

3) Volatility from lowest to highest

Then, your program should prompt the user for the name of the file the user would like to store the output.

You must create a stock struct for your solution and write appropriate functions. (Points will be taken off for correct programs that don't use a struct or reasonable functions.)

Defintion of Change

Given a list of prices for a share of a stock: s_1, s_2, s_3, …, s_n for days 1 through n, we define the change of the value of the stock over the interval to simply be $Delta(S) = s_n - s_1$.

Definition of Average Price

Given a list of prices for a share of a stock: s_1, s_2, s_3, …, s_n for days 1 through n, we define the average price of the stock to be $Avg(S) = \frac{\sum_{i=1}^{n} s_i}{n}$. (This is just the sum of each stock price in the interval divided by the number of prices.)

Definition of Volatility

Given a list of prices for a share of a stock: s_1, s_2, s_3, …, s_n for days 1 through n, we define the volatility of the stock to be $\frac{100 \sum_{i=1}^{n-1} |s_{i+1} - s_i|}{Avg(S)}$. (This is just the sum the change in the stock price between each consecutive day times 100 divided by the average of the stock prices.)

Input File Format

The first line of the input file will contain a single integer, *n*, representing the number of different stocks for which there are data. Each stock record follows, one per line. Each of these lines begins with a string of 5 or fewer characters known as the stock ticker, representing the stock. This is followed by a space and another 20 space separated floating point numbers, representing the price of that stock on 20 consecutive days.

Output File Format

For each of the different queries, follow the same output format. On the first line, output a single integer, *n*, representing the number of stocks in the file. On the following *n* lines, list the stocks in the desired order, listing the stock ticker, followed by a space, followed by the appropriate measure (change, average or volatility) printed to 2 decimal places. If there are ties, you may print tied items in any order. (So, there may be more than 1 possible correct output.)

Sample Program Run

```
The stock data has been loaded.
How would you like to sort the data?
1) change (highest positive change first)
2) average value (highest average first)
3) volatility (lowest volatility first)
3
What file would you like the output to be stored?
volatile.txt
```

Sample Data

This will be provided online soon.

Implementation Requirements (for full credit)

1) Must use a struct.
2) Must write several functions besides main.
3) Code must be follow good style and be well-commented.
4) Output must be correct.

Restrictions

Although you may use other compilers, your program must compile and run using gcc in Code::Blocks. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. You should also include comments throughout your code, when appropriate. If you have any questions about this, please see a TA. **Your program must read its input from stockdata.txt.**

Deliverables

A single source file named *stocks.c* turned in through WebCourses.

Fall 2013 Section 4 COP 3223 Program #12 Grading Criteria

Stock Sorting (100 pts total)

Coding Style Points (16 pts)

Header Comment – 2 pt
Internal Comments – 4 pts
Good variable names – 2 pt
Indented properly – 2 pt
Use of white space – 2 pt
Consistency of style – 2 pt
Uses constants – 2 pt

Code Points (24 pts)

Has required I/O (2 pts)
Uses a struct to store a stock (2 pts)
Declares an array of that struct (2 pts)
Attempts to calculate change for a single stock (2 pts)
Attempts to calculate average for a single stock (2 pts)
Attempts to calculate volatility for a single stock (2 pts)
Attempts to sort (4 pts)
Has a reasonable function breakdown – 8 pts (you decide partial)

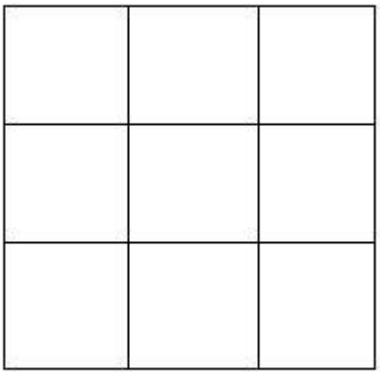
Execution Points(60 pts) - none can be awarded if the file doesn't compile (But double check if error is really simple, in that case, you probably have a different compiler...just fix the slight error and move on.)

2 test files: stockdata.txt, stockdata2.txt, 10 points for each output file, do NOT take off any points for rounding errors that are off by no more than .1. You may give partial credit on a file out of 10 points if some of the answers are correct.

COP 3223 Program #4: Turtle Time and List Power

Part A: Turtle Tac Toe (ttt.py)

Hopefully you enjoyed the turtle videos. For this portion of the assignment, you'll write a short program that prints out an empty Tic-Tac-Toe grid, using python's turtle. You can scale the grid to be the size that you want, but make sure that its shape looks like this:



For a bit of extra credit, allow two uses to play tic tac toe against each other by prompting them which row and column they want to put their piece (row 0 – 2, col 0 – 2).

Part B: Grocery List (grocery.py)

Write a program where the user enters a grocery list and then enters what they have actually bought. Afterwards, print out a list of items left to obtain as well as a list of unnecessary items that they bought. Print both of these lists in alphabetical order. The two input lists can come in any order and may contain repeats. For example, if the initial list contains "eggs" five times but we only buy "eggs" two times, then our list of items left to buy should contain three copies of "eggs".

Sample Program Run

```
Please enter the number of items on your grocery list.
6
What is item #1 on your list?
eggs
What is item #2 on your list?
cheese
What is item #3 on your list?
milk
What is item #4 on your list?
ham
What is item #5 on your list?
eggs
```

```
What is item #6?
bread
Please enter the number of items you bought.
4
What is item #1 that you bought?
milk
What is item #2 that you bought?
chips
What is item #3 that you bought?
turkey
What is item #4 that you bought?
cheese
Here are the items you still need to buy:
bread
eggs
eggs
ham
Here are the unnecessary items you bought:
chips
turkey
```

Deliverables

Two source files: *ttt.py*, for your solution to problem A, *grocery.py* for your solution to problem B. All files are to be submitted over WebCourses.

Restrictions

Although you may use other compilers and coding environments, your program must run in IDLE.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

Fall 2013 Section 4 COP 3223 Program #4 Grading Criteria

Part A: Tic-Tac-Toe (40 pts)

Coding Style Points (10 pts)
Written in a separate file - 1 pt
Uses main function - 1 pt
Header comment - 2 pt
Constants – 1 pt
Internal comments - 2 pts
Good variable names - 2 pt
1 pt free =)

Code Points (10 pts)
Uses a loop (5 pts) – yes, people will get mad at me for having this here, it's much cleaner to use a loop and at this point it's important to reward people who can see repetition and how to express it in a programming language.

Uses reasonable turtle functions (5 pts)

Test Case (20 pts)
It works – give partial credit as you see fit…

Part B: Grocery Lists (60 pts)

Coding Style Points (10 pts)
Written in a separate file - 1 pt
Uses main function - 1 pt
Good use of whitespace – 1 pt
Header comment - 2 pts
Internal comments - 3 pt
Good variable names - 2 pt

Code Points (20 pts)
Reads in all input (5 pts)
Uses two lists (5 pts)
Sorts lists (5 pts)
Prints out two lists (5 pts)

Test Cases (30 pts)
3 test cases (you can make them up, but make sure of testing these three categories)

Case 1: Add 7 unique items, removing 3 of them
Case 2: Add 6 items with some repeats, remove 4 items – with 2 on the list, 2 not on
Case 3: Add 10 items (2a, 3b, c, d, e, f, g), Remove (3a, 2b, e, h, i)
Give partial as necessary.

COP 3223 Program #1: Vacation Planning

Due date: Please consult WebCourses for your section

Notes

1. Please read the notes on Code::Blocks provided on the course web page.

Objectives

1. To give students practice at typing in, compiling and running simple programs.
2. To learn how to read in input from the user.
3. To learn how to use assignment statements and arithmetic expressions to make calculations.

Introduction: Vacations

Arup was tardy getting together the class because of a planned vacation (to attend a wedding). Unfortunately, as Arup found out, traveling with a full family (as opposed to just traveling alone), can get expensive. In this assignment you will write four separate programs in all. The first three will calculate a family's travel, food and lodging costs while the fourth will put together all of the first three programs to calculate a total vacation cost.

Part A: Travel Costs (travel.py)

Each vacation involves some travel costs. Typically, these include flying and renting a car. Write a program to calculate the travel costs of a family. Your program should prompt the user for the following information:

1) The number of people in the family
2) The number of days spent on vacation
3) The cost of a single plane ticket (round trip) in dollars

For the purposes of this problem, assume that the cost of the rental car per day is simply $20/per person. (Thus, a rental car big enough for 3 people would cost $60/day.)

Input Specification

All three input values will be positive integers.

Output Specification

Output a single line with the following format:

```
Your family will spend $X on travel costs for your vacation.
```

where X represents the total travel costs for the specified vacation. Do not worry about the number of digits that print after the decimal.

Sample Program Run (User Input in Bold)

```
How many people are in the family?
4
How many days will your family be on vacation?
3
What is the cost of a plane ticket, in dollars?
200
Your family will spend $1040 on travel costs for your vacation.
```

Part B: Food Costs (food.py)

Unfortunately, when on vacation, a family must typically eat out. In most situations, breakfast and lunch are eaten at fast-food restaurants that require no tip while dinner is eaten at a sit down restaurant that requires a tip. Write a program to calculate the food costs of a family. Your program should prompt the user for the following information:

1) The number of people in the family.
2) The number of days spent on vacation (assume full days with 3 meals/day)
3) The sales tax in the local area of the vacation, as a percentage.

In order to make your calculation, use the following constants:

```
BKF_COST_PERSON = 5
LNC_COST_PERSON = 8
DIN_COST_PERSON = 13
TIP_PERC = 18
```

Note: The first three are in dollars and the last is a percentage.

Input Specification

The first two values will be positive integers while the last will be a positive real number less than 20.

Output Specification

Output a single line with the following format:

```
Your family will spend $X for food on your vacation.
```

where X represents the total food costs for the specified vacation. Do not worry about the number of digits that print after the decimal.

Sample Program Run (User Input in Bold)

```
How many people are in the family?
4
How many days will your family be on vacation?
3
What is the sales tax percentage in the vacation locale?
6.5
Your family will spend $362.19 for food on your vacation.
```

Part C: Hotel Costs (hotel.py)

Last, but not least, there are typically hotel costs when a family travels. For this program, compute the cost of a family staying in a hotel. Your program should prompt the user to enter the following information:

1) The number of people in the family.
2) The number of days spent on vacation (assume full days with 3 meals/day)
3) The sales tax in the local area of the vacation, as a percentage.

Assume that a full hotel room houses 4 people and costs $100 per night. For smaller hotel rooms, there's a per person charge of $30. (Thus, for a family of 10, two full rooms would fit 8 people, costing $200/night and a third room would have 2 people, costing $60/night for a grand total of $260/night for the family.)

Input Specification

The first two values will be positive integers while the last will be a positive real number less than 20.

Output Specification

Output a single line with the following format:

```
Your family will spend $X for lodging on your vacation.
```

where X represents the total hotel costs for the specified vacation. Do not worry about the number of digits that print after the decimal.

Sample Program Run (User Input in Bold)

```
How many people are in the family?
5
How many days will your family be on vacation?
3
What is the sales tax percentage in the vacation locale?
6.5
Your family will spend $415.35 for lodging on your vacation.
```

Part D: Total Cost (vacation.py)

Combine your three programs into one which prompts the user to enter 4 pieces of information and calculates the total cost of the whole vacation, which is the sum of the travel, food and hotel costs.

Sample Program Run (User Input in Bold)

```
How many people are in the family?
4
How many days will your family be on vacation?
3
What is the cost of a plane ticket, in dollars?
200
What is the sales tax percentage in the vacation locale?
6.5
Your family will spend $1721.69 in total for your vacation.
```

Deliverables
Four source files:

1) *travel.py*, for your solution to problem A
2) *food.py* for your solution to problem B
3) *hotel.py* for your solution to problem C
4) *vacation.py* for your solution to problem D

All files are to be submitted over WebCourses.

Restrictions
Although you may use other compilers and coding environments, your program must run in IDLE.

Grading Details
Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

Fall 2013 Section 4 COP 3223 Program #1 Grading Criteria

1 pt for each program for turning a non-empty .py file in

Coding Style Points (6 pts) – for all four programs, apply to each
Written in a separate file - 1 pt
Uses main function - 1 pt
Header comment - 1 pt
Internal comments - 1 pt
Declare constants (either beginning of main or before main) - 1 pt
Good variable names - 1 pt

Code Points (6 pts) – for all four programs
Reads in required data – 2 pts
Attempts calculations with data – 2 pts
Prints out some result – 2 pts

Test cases Part A (4 pts per case)
1) 10, 20, 225 (Answer = $6250)
2) 1, 1, 1, (Answer = $21)

Test cases Part B (4 pts per case)
1) 1, 1, 1 (Answer = $28.62, don't worry about # of decimals)
2) 17, 13, 8.2 (Answer = $6776.72, don't worry about # of decimals)

Test cases Part C (4 pts per case)
1) 1, 4, 6.5 (Answer = $127.80, don't worry about # of decimals)
2) 4, 1, 6.5 (Answer = $106.50, don't worry about # of decimals)
3) 6, 3, 7 (Answer = $513.60, don't worry about # of decimals)
4) 3, 5, 5 (Answer = $472.50, don't worry about # of decimals)
5) 75, 50, 8.5 (Answer = $102532.50, don't worry about # of decimals)

Test cases Part D (4 pts per case)
1) 1, 1, 200, 6.5 (Answer = $282.13, don't worry about # of decimals)
2) 4, 5 250, 7 (Answer = $2541.48, don't worry about # of decimals)
3) 14, 23, 289, 8.25 (Answer = $29327.43, don't worry about # of decimals)