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%.**