Lab 7: Programming Competition

CSE/IT 107

NMT Computer Science

"First, solve the problem. Then, write the code."

— John Johnson

"Beware of bugs in the above code; I have only proved it correct, not tried it."

— Donald Knuth

"Openness always deserves recognition." ("Offenheit verdient immer Anerkennung.")

— Otto von Bismarck

---

1 Introduction

This lab is solely a problem solving lab.

2 Programming Competition

Please pair up into groups of two or three with somebody in your lab. You will enter into the competition together and submit the same files.

To submit your answers, please notify a TA. The whiteboard will serve as the scoreboard for your lab. The TA will want to see the answer to the "scoreboard submission." If the TAs are grading another team's submission, do not wait for them and move on to a new problem.

Do not share the problem descriptions or the code with the other lab section. Everyone needs to have an equally fair chance of doing well in this competition.

You may not use the internet to find answers to these problems. Think! If the concepts are unclear, please get help from a TA.

2.1 Extra Credit

This lab will not count for its own grade. Your standing in the competition will count for extra credit on your lab final. The winners of your lab section will get 10%, the second place will get 9%, and so on as extra credit on their lab final.

You have to gain at least 20 points in the competition to qualify for the extra credit.

2.2 Requirements

Even though this lab "only" counts for extra credit, you have to stay the lab and try your best. **You will LOSE 10% on your lab final if you do not stay and try your best.**

You may choose to do any of the following problems. However, there is no credit for incomplete or incorrect submissions.

2.3 Problems

Problem [1] **Multiples of 3 and 5** (5 pts, filename: `multiples.py`)

If we list all the integers (natural numbers) below 10 that are multiples of 3 or 5, we get 3, 5, 6, and 9. The sum of these multiples is 23.

Write a program to find the sum of all the multiples of 3 or 5 below 1000.

Scoreboard submission: The sum of all multiples of 3 or 5 below 1000.

Problem [2] **Special Pythagorean Triplet** (10 pts, filename: `pythagoras.py`)

A Pythagorean triplet is a set of three integers (natural numbers), $a < b < c$ such that $a^2 + b^2 = c^2$.

For example, $a = 3, b = 4, c = 5$ fits: $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

There exists exactly one Pythagorean triplet for which $a + b + c = 1000$.

Scoreboard submission: Find the product *abc* of that special Pythagorean triplet.

Problem [3] **Powers** (10 pts, filename: `powers.py`)

Write a program that determines whether a given integer is a power of another integer and what the base and exponent are; for example, $32 = 2^5$. Here, 2 is the base and 5 is the exponent. You may assume that the integer given is less than $1,000,000$.

For example:

```
A number: 32
32 is 2 to the 5th power.

A number: 25
25 is 5 to the 2nd power.

A number: 23
23 is not a power of any number.
```

Scoreboard submission: Find an integer base and exponent for 823,543.

Problem [4] **Fractions** (15 pts, filename: `fractions.py`)

Write a program that reads both the numerator and denominator of a fraction and displays the reduced fraction. For simplicity, you may assume that the numbers entered are less than 10,000.

For example:

```
Enter numerator: 4
Enter denominator: 30
Result: 2/15
```

```
Enter numerator: 25
Enter denominator: 5
Result: 5/1
```

Hints: See Appendix A to find out what a GCD is and what Euclid's algorithm is.

Scoreboard submission: There is none. You have to get this one looked at and tested by a TA.

Problem [5] **Coins** (20 pts, filename: `coins.py`)

Write a program that determines the minimum amount of (dollar) coins and bills to be used to change a certain amount of money. That is, find the largest number of 100s, then 50s, then 20s, then 10s, etc for the change. Assume that you are changing less than $500. Assume that you are using the following denominations:

- Bills: 100, 50, 20, 10, 5, 2, 1 dollars
- Coins: 25, 10, 5, 1 cents

For example:

```
Enter an amount of money (dollars): 10.33
Change:
10 dollar bill: 1
25 cent coin: 1
5 cent coin: 1
1 cent coin: 3
```

Scoreboard submission: Find the optimal amount of change for $76.23.

Problem [6] **Crossing Hands** (25 pts, filename: `hands.py`)

The hands of an analog clock occassionally cross as they revolve around the dial. Write a program that determines how many times the hands cross in one twelve-hour period, and compute a list of those times (down to seconds). Assume you are just using two hands: the hour hand and the minute hand. Also assume that they tick continuously.

Scoreboard submission Add up the hours, minutes, and seconds of the times separately and submit it in the format `H-M-S`, for example `30-167-421`.

Problem [7] **Block Areas** (35 pts, filename: `blocks.py`)

Write a program to take in a series of coordinate points and finds the enclosed area. The points will all be non-negative integers not exceeding 100. All angles in the shape will be 90 degrees. You can assume that no shape will be defined by more than 20 points.

For example:

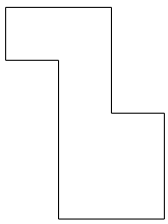

Figure 1: Example shape

```
Enter number of points: 8
Enter point: 2 1
Enter point: 4 1
Enter point: 4 3
```

```
Enter point: 3 3
Enter point: 3 5
Enter point: 1 5
Enter point: 1 4
Enter point: 2 4

Area: 7
```

Scoreboard submission: Find the area of the polygon described by the following 14 coordinates. The order of entering them should not matter!

$$(0,0)\ (6,0)\ (6,1)\ (2,1)\ (0,2)\ (2,2)\ (7,2)$$

$$(1,2)\ (1,3)\ (2,3)\ (2,4)\ (9,4)\ (9,1)\ (7,1)$$

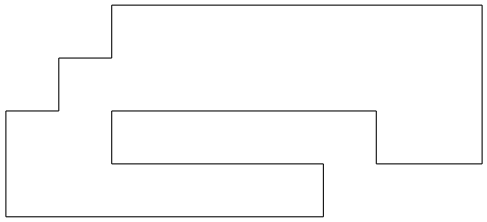

Figure 2: Scoreboard submission shape

Appendices

A Reducing Fractions – GCD and Euler's Algorithm

Reducing a fraction involves dividing the numerator and denominator by the same integers until there is nothing that divides both numerator and denominator anymore.

For example, $\frac{60}{6} = \frac{30}{3} = \frac{10}{1} = 10$. First, I divided both numerator and denominator by 2, and then by 3.

This is where the concept of a GCD comes in. GCD stands for "greatest common divisor" – given two numbers such as 60 and 6, the GCD is the greatest whole number that both numbers are divisible by. Since both 60 and 6 are divisible by 6 and since there is no greater integer than 6 that one can divide them both by, the $\gcd(60, 6) = 6$. For another example, $\gcd(40, 6) = 2$.

If you think about it more, it becomes clear that one can reduce a fraction by dividing both the numerator and denominator by the GCD of both. In essence, given a fraction $\frac{a}{b}$, the reduced fraction would be:

$$\frac{a}{b} = \frac{\frac{a}{\gcd(a,b)}}{\frac{b}{\gcd(a,b)}}$$

How do we find the GCD though? Here, a neat algorithm called Euclid's algorithm comes in. Euclid's algorithm takes advantage of the fact that when an integer a is divided by another integer b, we get a quotient q and a remainder r. This is called the Division Algorithm (see Theorem 1).

Theorem 1 (Division Algorithm)**.** *Given integers a and b, where $b > 0$, there exist unique integers q (the quotient) and r (the remainder) such that*

$$a = bq + r, \text{ where } 0 \leq r < b.$$

For example, when 5 is divided by 2, we get a quotient of 2 and a remainder of 1, since $5 = 2 \times 2 + 1$. You have already taken advantage of this algorithm when you validated credit card numbers.

In C, we can find the quotient by doing integer division and the remainder using the modulus operator %.

A.1 Euclid's Algorithm

Euclid's algorithm finds the GCD by repeatedly using the division algorithm. Given integers a and b, where $a > b$, that we want to find the GCD of, we find the quotient q and r. If the remainder is 0, b is the GCD. If the remainder is not 0, we repeat the same steps, but we let $a = b$ and $b = r$.

In essence, it goes:

$$\begin{aligned} a &= bq + r && \text{where } 0 \leq r < b \\ b &= rq_2 + r_2 && \text{where } 0 \leq r_2 < r \\ r &= r_2q_3 + r_3 && \text{where } 0 \leq r_3 < r_2 \end{aligned}$$

So we are saying that $\gcd(a, b) = \gcd(b, r) = \gcd(r, r_2) = \cdots = \gcd(r_n, 0) = r_n$. Notice that the quotient is not used at all, only the given integers a, b and the remainder r matter.

Notice that $r_n < \cdots < r_3 < r_2 < r < b$.

It seems natural to write this with recursion. In pseudocode, with recursion:

```
1  euclids(a, b)
2          if(a < b)
3                  return euclids(b, a)
4
5          r = a modulus b
6          if(r == 0)
7                  return b
8          else
9                  return euclids(b, r)
```