

CS 102: Fundamentals of Computation

Spring 2021 – Midterm 2

03/31/21 – 7:00 p.m. - 9:00 p.m.
110 min + 10 min to scan/upload = 120 min

[Complete all the information in the box below.]

Name: _____

Student ID: _____

Email: _____@usc.edu

Lecture section (Circle One):

Redekopp
~~10 a.m.~~

Redekopp
~~11 a.m.~~

Ques.	Your score	Max score	Recommended Time
1		18	40 min.
2		18	35 min.
3		14	35 min.
Total		50	110 min.

Each of the 3 problems is described below and at the end of each description is a link/URL to a skeleton file in which you can write your code.

We prefer you upload the finished .cpp files but you may handwrite/annotate the pages below and submit a scanned PDF.

1. **Reduced Fractions (18 points):** Write a program that takes in **2 positive integers** representing the **numerator** and **denominator** of a fraction (i.e. $\frac{num}{denom}$), along with a single character **m** (mixed number) or **f** (fraction), and then reduces the fraction to simplest form (using the `gcd()` function we provide) and then outputs:
1. the decimal equivalent of the fraction (e.g. 1.333 for an input of 4 3 => 4/3) accurate to **3 decimal places, and**
 2. If the user entered **m**, output a mixed fraction, such as 1 1/3 (i.e. a whole part (if it's nonzero) followed by a fractional part, reduced).
 3. If the user entered **f**, output an improper fraction, such as 4/3 (reduced).
 4. A fraction that reduces to an **integer** should skip these steps and **just output the decimal form** from (1).
 5. Note that fractions that evaluate to a **value less than 1** should look the same for both types. (Do not print a whole part equal to 0)

Example 1: The input below specifies **20/5** (= 4/1 when reduced). Since the result is an integer, we only show the decimal equivalent **and nothing more**.

Input:

Output:

Example 2: The input below specifies **20/8** (= 5/2 when reduced). Since the result is not an integer and they entered **f**, we show the **decimal equivalent** and the **reduced irregular fraction**.

Input:

Output:

Example 3: The input below specifies **20/8** (= 5/2 when reduced). Since the result is not an integer and they entered **m**, we show the **decimal equivalent** and the **mixed number form**.

Input:

Output:

Example 4: The input below specifies **8/20** (= 2/5 when reduced). Since the result is not an integer and they entered m, we show the **decimal equivalent** and the **mixed number form** but **without the leading integer 0**.

Input:

Output:

Decimal: 0.400 Reduced fraction: 2/5

Requirements and Assumptions

- You may assume the user will always input **positive** integers and will give correct, expected inputs (i.e. 2 positive integers followed by **m** or **f**)
- You may assume the denominator will **NOT** be 0
- You must output the decimal equivalent rounded to **exactly 3 decimal places**.
- We have provided a function in the skeleton code: `int gcd(int x, int y)` that will return the **greatest common divisor/multiple** of the 2 input arguments. It is written at the bottom of the skeleton file and is correct. By calling this function, you can use its return value to find the simplest (reduced) form of the fraction. For example, `gcd(6, 15)` would return 3. With a little thought you should be able to figure out how to use that return value (i.e. 3) to reduce the fraction (e.g. reduce 6/15 to the desired 2/5).
- Your output should match the format shown in the examples below.

The skeleton file on the next page can be downloaded with the link: reduce.cpp .
--

```

#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

int gcd(int x, int y); // prototype of working GCD function

int main()
{
    int num, denom; // numerator and denominator, respectively
    char fmt;        // character to store 'f' or 'm'
    //Declare other variables and write your code below. If you cannot figure out how to
    //reduce the number, just perform the rest of the task using the non-reduced fraction.


    return 0;
}
// working implementation to find and return the gcd of 2 numbers, x and y
int gcd(int x, int y)
{
    while(y != 0){ int t = y;    y = x % y;    x = t; }
    return x;
}

```

2. **Vaccine Doses (18 pts).** Each day, a **new number of people sign up to get vaccinated** and a **new number of doses of vaccine arrive**. Write a program that takes as input **two integers at a time** representing the **number of new people that want a vaccine dose** and the **number of new vaccine doses that arrive**, respectively. Then, for each day, output how many people can get a dose of the vaccine (based on supply). If at some point, there are more people who want a vaccine dose than doses available, the number of people still wanting a dose should carry over from day to day until there is an available dose. In the same way, if there are more doses available than people who want them, the extra doses should carry over day to day to be used in the future (assume they do not expire).

Keep taking in pairs of **integers** representing the **new number of people wanting doses** and the **new vaccine doses available** on that day **until** one or the other integer values is **negative**, at which point the program should quit.

For each pair of positive integers input, output **the day number** (starting from 1 and increasing 1 day at a time until the program quits) and **how many people were able to be vaccinated that day** (i.e. including people from previous days when supply was insufficient).

Example: See the **example** immediately below and the **explanation** just below that.

```
How many new people and doses arrived today: 5 7
Day 1: 5 people got doses today
How many new people and doses arrived today: 4 1
Day 2: 3 people got doses today
How many new people and doses arrived today: 0 2
Day 3: 1 people got doses today
How many new people and doses arrived today: 0 4
Day 4: 0 people got doses today
How many new people and doses arrived today: 7 0
Day 5: 5 people got doses today
How many new people and doses arrived today: -1 1
```

Explanation:

- On day 1, 5 people wanted doses and 7 doses arrived, so all 5 people could get vaccinated and 2 doses are saved for the future.
- On day 2, 4 people wanted doses and 1 dose arrived, so using the 2 leftover doses from the previous day and the 1 that arrived today, 3 of the 4 people could get vaccinated. The other person will wait for supply in the coming day(s).
- On day 3, 0 new people wanted doses and 2 doses arrived, so the 1 person from yesterday got a vaccine and the 1 other dose is saved for the future.
- On day 4, 0 new people wanted doses and 4 doses arrived, so 0 people got vaccinated and the 1 dose from yesterday plus the 4 doses from today are saved for the future.
- On day 5, 7 new people wanted doses and 0 arrived, so 5 people got vaccinated using the 5 saved vaccine doses from previous days.
- On day 6, a negative number was input and thus the program quits.

Requirements

- You can assume the user will always input **2 integers per day (not just 1)**.
- Your prompt text and output message text should be similar to the example above.

- You must ensure you never give more people doses than are available and you must carry over all waiting people or surplus doses indefinitely (i.e. people don't stop wanting the vaccine nor do doses expire).

The skeleton below can be downloaded with the link: [doses.cpp](#).

```
#include <iostream>
#include <iomanip>
#include <algorithm>
using namespace std;
int main()
{
    int np, nd;    // number of people wanting doses and number of doses on a day
    int day=1;     // variable to track what day it is
    int wp=0, wd=0; // carry-over of people waiting and doses available
    // Declare other variables and complete your code below (changing what you want)
    cout << "How many new people and doses arrived today: ";
    cin >> np >> nd;

    return 0;
}
```

3. In this program, the user will input an **integer** starting length (len), and a single character **u** or **l** (the lowercase of **L**...NOT the number one) and then you will draw an **"arrow"** pattern to the screen using **lower case letters** (if the user entered **l**) or **upper case letters** (if the user entered **u**).

The first and last row should have **1en** (i.e. **20** in Example 1) lowercase character (since the user input **1**) printed on that row, starting at **a** and increasing to **b**, **c**, etc. The second and second to last row should have **1en+1** consecutive characters (starting at **a**) printed on those two rows, and so on until the middle row which has **all 26 lowercase letters printed**.

If the length input is **15 u** (as in Example 2) the first and last row output should have **15** consecutive **uppercase** letters starting with **A** and then continue row by row until the middle row has all **26** uppercase letters.

Example 1 and 2:

[illegible]

Example 3: If the length input is **26 1**, the only row that should be output should have all **26 lowercase** letters.

abcdefghijklmnopqrstuvwxyz

Requirements and Assumptions

1. If the user inputs a length that is **NOT** in the range **1-26**, simply **default to a length to 20**.
2. You may assume the 2nd input from the user is always **l** or **u** (no need to check for any other character).
3. You may **NOT** use the **string** type.
4. You do not need arrays to solve this problem.

The skeleton on the next page can be downloaded with the link: [arrow.cpp](#).

```
#include <iostream>    // You may not use the string type
using namespace std;
int main()
{    // Declare your variables and write your code below.
    // You may not use the string type

    return 0;
}
```