Assignment 1 - Part 2

- Must be done individually. See website for due date and late penalty.
- Submit all deliverables to the CourSys: https://coursys.sfu.ca/
- Do not show another student your code, do not copy code found online, and do not post questions about the assignment online. Please post all questions to the Piazza forum (either as a public or private message).
- See the marking guide for details on how each part will be marked. Variable names, commenting, and code indentation are always important.

1. Big Popcorn Party

Your friend is selling popcorn as a fundraiser for a school. They need your help figuring out how much popcorn to buy. They are going to buy pre-popped popcorn from a local movie theatre in 50L bags. They are going to serve each person 2-cups of popcorn. They need to know, for a given number of people:

1. How many giant 50L bags of popcorn do they need to buy?
   Note that the theatre only sells whole number bags of popcorn; you cannot buy 3.2532352 bags of popcorn! You must buy 4 in this case to feed everyone.

2. How many 2 cup servings can they get out of the popcorn they are buying?
   This is interesting because they have to buy a whole number of giant bags of popcorn, and hence they are likely to have more than they need.

Design and write your C++ program in a file named `popcorn.cpp`. Ask the user to enter how many people are coming to the fundraiser and then print the answers for the above two questions.

Your output must be formatted as shown in the sample output, complete with the double quoted statement to tell your friend. Test your program to ensure it works with inputs like 1, 10, 100, 0. You don't need to worry about handling a negative number, or a number of people over 100,000.

- See the sample output below for required formatting. Bold/underlined text is only shown here to indicate it's inputted by the user and not part of the actual output.
- The line of text for what to tell your friend is all one line.
- When this program finishes (as with all labs/assignments), it must print a line-feed after it last output (i.e. your last `cout` should still have an `endl` or "\n")
- Assume there are 4.22675 cups per litre.

- *Hint:* Choose your variable data types carefully. If you expect the number to be a floating point (like 2.52), ensure it's stored in a `double`. If it's a whole number, use an `int`.
- *Hint*: Given a floating point number (such as stored in a `double`), you can round it *up* using `ceil()`, round it *down* using `floor()`, and round it *to the nearest whole number* using `round()`. These functions are in the `cmath` file, so there's a `#include` needed. Here is a full sample program using these:

```
// Round a number up, down, and to the nearest whole number
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  double myNumber = 3.14159;
  cout << "Rounded up is " << ceil(myNumber) << endl;
  cout << "Rounded down is " << floor(myNumber) << endl;
  cout << "Rounded to the nearest " << round(myNumber) << endl;
  return 0;
}
```

Sample Output

```
How many students are expected at the event? 1
Tell your friend, "You need 0.00946354 mega bags of popcorn, so order
1 bag(s) total!"
This gives you enough popcorn to serve 105 students.
```

Output 1: Sample output.

```
How many students are expected at the event? 15
Tell your friend, "You need 0.141953 mega bags of popcorn, so order 1
bag(s) total!"
This gives you enough popcorn to serve 105 students.
```

Output 2: Sample output.

```
How many students are expected at the event? 100000
Tell your friend, "You need 946.354 mega bags of popcorn, so order 947
bag(s) total!"
This gives you enough popcorn to serve 100068 students.
```

Output 3: Sample output.

2. Fruit Box Calculator

Create a C++ program named `fruitbasket.cpp` which helps create fruit baskets from donated apples and oranges. Its purpose is to calculate the number of baskets that can be created.

Try to make all baskets as similar as possible. For example, if each basket requires 4 pieces of fruit and there are 25 apples and 16 oranges then:

- 41 pieces of fruit creates 10 baskets with one piece of fruit left over.
- Each basket should get at least 2 apples and 1 orange, plus one one more piece of fruit (either an apple or an orange).

Requirements

- The program asks the user to enter (via the keyboard) three integers:
  - The number of apples donated.
  - The number of oranges donated.
  - The required number of fruit pieces per basket.
- Assume all input values are integers between 0 and 1,000,000.
  - You need **not** check for invalid values such as negative numbers, too big numbers (like 2 million), fractional numbers (like 3.1), negative numbers (like -5), or non-numeric values (like “hello world”).
  - Assume there is enough fruit to make at least one basket. (Otherwise, your program may crash, and we don't yet know enough C++ to handle this yet!)
- Calculate and display the following:
  - The total number of baskets possible to create using the donated fruit, each containing exactly the required number of pieces of fruit.
  - Minimum number of apples per basket. This is, if the apples were split evenly among the baskets, at *least* how many apples are there per basket?[1]
  - Minimum number of oranges per basket (same as above).
  - Once created, each basket will be tied off with three pieces of ribbon per basket. Calculate the total number of pieces of ribbon required.
  - Number of pieces of fruit required to make one more basket.
- Your program's output should look like the samples shown below.
  - Note that data must lineup, as shown in the sample outputs.
  - Format to handle numbers up to 4 digits wide (as shown in the outputs). This should right align the values.

1 Note that this turns out *not* to be “at least how many apples are there per basket” because in some cases the minimum number of apples + minimum number of oranges could be more than a basket requires! So, we are more accurately calculating: “There are at least this many apples available to put in each basket, even if all baskets turn out not to need this many.”

For example, given 10 apples and 10 oranges and making baskets of 15 fruit. There are 20 fruit, so we can make 1 basket. This means there are 10 apples available for this basket (“Min # apples per basket” = 10), and there are 10 oranges available for this basket (“Min # oranges per basket” = 10). But, if that basket has 10 apples and 10 oranges, it means there would be 20 fruit! So, instead this really means that there are at least 10 apples and 10 oranges *available* for each basket, even though not going to need them all.


```
*************************
Local Fruit Box Calculator
*************************

Enter the number of apples donated: 25
Enter the number of oranges donated: 16
Enter the number of fruit pieces per basket: 4

Input Values:
-------------
# Apples Donated:            25
# Oranges Donated:           16
# Fruit Pieces per Basket:    4

Basket Creation Numbers:
-----------------------
# Baskets to create:        10
Min # apples per basket:     2
Min # oranges per basket:    1
# Ribbons to tie baskets:   30
# Pieces needed to complete one more basket:    3
```

Output 4: Sample output.

```
*************************
Local Fruit Box Calculator
*************************
Enter the number of apples donated: 2502
Enter the number of oranges donated: 9103
Enter the number of fruit pieces per basket: 5

Input Values:
-------------
# Apples Donated:          2502
# Oranges Donated:         9103
# Fruit Pieces per Basket:    5

Basket Creation Numbers:
-----------------------
# Baskets to create:      2321
Min # apples per basket:     1
Min # oranges per basket:    3
# Ribbons to tie baskets: 6963
# Pieces needed to complete one more basket:    5
```

Output 5: Sample output.


```
*************************
Local Fruit Box Calculator
*************************
Enter the number of apples donated: 100
Enter the number of oranges donated: 123456
Enter the number of fruit pieces per basket: 10

Input Values:
-------------
# Apples Donated:           100
# Oranges Donated:         123456
# Fruit Pieces per Basket:   10

Basket Creation Numbers:
------------------------
# Baskets to create:      12355
Min # apples per basket:     0
Min # oranges per basket:    9
# Ribbons to tie baskets: 37065
# Pieces needed to complete one more basket:    4
```

Output 6: Sample output showing input size larger than expected (alignment problems). Your solution should also look like this.

```
*************************
Local Fruit Box Calculator
*************************
Enter the number of apples donated: 0
Enter the number of oranges donated: 5
Enter the number of fruit pieces per basket: 10
Floating point exception (core dumped)
```

*Output 7: Sample output showing your program **may** crash when it cannot make even one basket. We will not test for this condition because we don't yet know how to handle it. (The last line of the output is generated by the OS when it detects that the program crashed due to a "floating point exception".)*

3. Deliverables

Submit the items listed below to the CourSys: https://coursys.sfu.ca/
(Note you do not have to create a zip file for these; CourSys will accept them directly. However, your files must be named correctly.)

- Part 1's paragraphs as a .PDF file.
- `popcorn.cpp`
- `fruitbasket.cpp`

Submit all files at the same time. You can re-submit any number of times; only your last submission will be marked. If changing only one of your files, please still submit all files together. If you only submit one file then CourSys will not show us the other files when we mark.

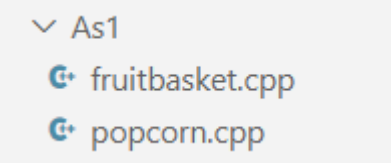
All submissions will be compared for unexplainable similarities. We expect submissions will be somewhat similar for this assignment but do your own original work; we are checking!

Do not email/give your code to another student. Do not accept code from another student. Do not post your code online.

3.1 Troubleshooting Submission Problem

Here are some things to look at if you have challenges submitting:

1. When submitting your source code to CourSys, it will enforce that the files have the correct names. Ensure your file names are exactly as expected, and not "`popcorn1.cpp`" or "`popcron.cpp`" (typo).
2. Ensure your file names are the correct case (lower case), and not "`Popcorn.cpp`" or "`popcorn.CPP`".
3. Ensure your files have a `.cpp` file extension. Double check you are not trying to submit the compiled executable "`popcorn`"; you need to submit "`popcorn.cpp`"
4. Ensure your files do not have anything extra in their name, such as "`popcorn.cpp.source`". Your computer may hide file extensions by default, so search the web for "Show file extensions in ___" (fill in the blank with your OS) for directions.
5. Ensure that you have named *C++ files* as required, as opposed to having named the *folder* with that name. For example, VS Code's explorer pane should look like the following (OK to name your folder anything you like, or for other files to be in the folder as well).



6. After submitting, you can verify if any files were submitted by viewing the assignment page on CourSys. It should allow you to view/download your submission.

