CS215: Introduction to Program Design, Abstraction and Problem Solving
(Spring, 2025)
Lab Assignment 2
(20 points)
Today's Date: Tuesday, January 20
Demonstration Due Date: the end of Lab3 class
Submission Due Date: Friday, January 31

The purpose of this lab assignment is

- to get familiar with Microsoft Visual Studio IDE.
- to practice using input/output.
- to practice using variables, constants, fundamental data types and basic operations.

Problem specification

Write a program that helps a bank teller to provide currency exchange service at the airport. The program has one input: the amount received from the customer (expressed in Canadian Dollars (CAD)). Display the dollars, dimes, nickels, and pennies that the customer should receive in US Dollars (USD), based on the exchange rate:

```
1 CAD = 0.69 USD.
```

The program collects one user input: the amount of cash provided by the customer (expressed in CAD). Display the dollars, dimes, nickels, and pennies (expressed in USD) that the customer should receive through the exchange service .

The following are sample outputs of running your program SIX times: (Note that the blue part represents the user input, and "↲" represents the enter/return key from the user input)

Sample output 1:

```
Convert CAD to USD service.
Please enter the amount of Canadian Dollars you want to
exchange: C$ 6.30↲
The exchange for C$6.30 --> $ 4.35 :
Dollars:              4
Quarters:             1
Dimes:                1
Nickels:              0
Pennies:              0
```

Sample output 2:

```
Convert CAD to USD service.
Please enter the amount of Canadian Dollars you want to
exchange: C$ 605.25↲
The exchange for C$605.25 --> $ 417.62 :
```

```
Dollars:            417
Quarters:             2
Dimes:                1
Nickels:              0
Pennies:              2
```

Sample output 3:

```
Convert CAD to USD service.
Please enter the amount of Canadian Dollars you want to
exchange: C$ 113.00↲
The exchange for C$113.00 --> $ 77.97 :
Dollars:             77
Quarters:             3
Dimes:                2
Nickels:              0
Pennies:              2
```

Sample output 4:

```
Convert CAD to USD service.
Please enter the amount of Canadian Dollars you want to
exchange: C$ 215.70↲
The exchange for C$215.70 --> $ 148.83 :
Dollars:            148
Quarters:             3
Dimes:                0
Nickels:              1
Pennies:              3
```

Sample output 5:

```
Convert CAD to USD service.
Please enter the amount of Canadian Dollars you want to
exchange: C$ 0.00↲
The exchange for C$0.00 --> $ 0.00 :
Dollars:              0
Quarters:             0
Dimes:                0
Nickels:              0
Pennies:              0
```

Sample output 6:

```
Convert CAD to USD service.
Please enter the amount of Canadian Dollars you want to
exchange: C$ -50.00↲
The exchange for C$-50.00 --> $ -34.50 :
Dollars:            -34
Quarters:            -1
Dimes:               -2
```

```
Nickels:              0
Pennies:             -4
```

Note that *Sample output 6* is the case, when the user input is a negative number. It will generate the negative numbers in the exchange from your program. However, it is NOT practical!!! In your program design, you may want to display the message to the customer that the amount for the exchange needs to be a positive number. For now, you can leave it as it is shown in "Sample output 6". After learning the conditional statements in Chapter 3, we will do better by making decisions under different conditions.

To collect the user input: please remember that the user needs to be prompted for what kind of data you are asking for. (Prompts are done by output statement). User-friendliness in I/O design is important for good programming.

For the formatted output, you may want to control over the output appearance: (1) you may add one or two "`\t`" (tab key) after the words such as "Dollars", "Quarters", and so on to line up the output; (2) set up the total width of six characters for the values displayed in the exchanged currency. If the value is not six characters, spaces are printed before the number.

Demonstration and Submission

1. Each Lab assignment needs to demonstrate to your TA to be graded. You can demonstrate Lab2 during Lab2 class (with possible bonus 3 points) or no later than the end of Lab3 class (this is the **demonstration deadline** for Lab2).
If you finish Lab2 assignment during Lab2 class, you may demonstrate your program to your TA and answer your TA's questions, you can get up to 3 extra points for this lab assignment. (Note you can also demonstrate your program to your TA during Lab3 class. However, any demonstration later than the end of the Lab2 class cannot get bonus 3 points.)
If you need extra time, you can continue working on Lab2 assignment after the Lab class and try to finish it before the next Lab class. Then demonstrate your Lab2 during Lab3 class.
If you do not demonstrate your code, even if you submit it in Canvas, you will receive a grade of 0!! *The TA may ask you to make some corrections. If so, make the corrections and demonstrate again…repeat until you have 100%!*
2. After the successful demonstration, submit the code in Canvas. Open the link to Course Canvas page (https://www.uky.edu/canvas), and log in to your account using your LinkBlue ID and password. Please submit your **source code in a .cpp file** through link "**`Lab 2`**".
Even if you successfully demonstrated it to the TA, if you do not submit it in Canvas by* the submission deadline*, you will receive a grade of 0!

Grading (20 points + Bonus 3 points)

1. Attend the lab session or have a documented excused absence. (**5 points**)
2. Demonstrate your program to your TA and submit it in Canvas. (**15 points**)
   - Include comments as specified in the lecture notes. (2 points).

- User-friendliness in I/O design. (2 points).
- Take the user input correctly (2 points).
- Generate the correct output, taking care of roundoff errors. (6 points).
- The formatted output satisfies requirement. (3 points)

Demonstrate your program to your TA and answer TA's questions during Lab class when the same Lab assignment is given. (Bonus 3 points)

Programming tips:

- **Do Not Use Magic Numbers**

A magic number is a numeric constant that appears in your code without explanation. For example,

```
total_volume = bottle * 2;
```

Why 2? Instead use a named constant to make the code self-ducomenting:

```
const double BOTTLE_VOLUME = 2;
total_voume = bottle * BOTTLE_VOLUME;
```

There is another reason for using named constants. Suppose circumstances change, and the bottle volume is now 1.5 liters. If you use a named constant, you make a single change, and you are done. Otherwise, you must look at every rule of 2 in your program and ponder whether it means a bottle volume, or something else. In a program that is more than a few pages long, that is incredibly tedious and error-prone.

- **Avoid Roundoff Errors**

Try the following block of statements:

```
double price = 4.35;
int cents = 100 * price;
          // Should be 100 * 4.35 = 435
cout << cents << endl;
          // Prints 434!
```

Of course, one hundred times 4.35 is 435, but the program prints 434.
Most computers represent numbers in the binary system. In the binary system, there is no exact representation for 4.35, just as there is no exact representation for 1/3 in the decimal system. The representation used by the computer is just a little less than 4.35, so 100 times that value is just a little less than 435. When a floating-point value is converted to an integer, the entire fractional part, which is almost 1, is thrown away, and the integer 434 is stored in cents.
The remedy is to add 0.5 in order to round to the nearest integer:

```
int cents = 100 * price + 0.5;
```

- **Handle possible "information loss" warning message**

The above remedy avoids possible roundoff errors, however, when you compile your program, it issues a warning message: there is the risk of information loss. Occasionally you need to store a value into a variable of a different type, this may happen. When you do want to convert a floating-point value into an integer value and you know to be safe in a particular circumstance, such as converting dollar bills into cents, you can turn off the warning message by using a **cast**. You express a cast in C++ as follows:

```
int cents = static_cast<int>(100 * price + 0.5);
```

And you can do even better without magic numbers.