CS215: Introduction to Program Design, Abstraction and Problem Solving (Spring, 2023)
Lab Assignment 1
(20 points)

Today's Date: Sunday, January 15

Demonstration Due Date: the end of Lab2 class

Submission Due Date: Friday, January 27

The purpose of this lab assignment is

- to get familiar with Microsoft Visual Studio IDE.
- to compile and run your first C++ program.
- to follow Coding Style Guide criteria when writing code.
- to practice using input/output.
- to practice using variables, constants, fundamental data types and basic operations.
- to get familiar with assignment submission through Canvas.

Problem specification

If you are going to work on computers in Lab classroom or you have already installed Visual Studio Enterprise 2022 on your own laptop, you can skip **Part 0**.

Part 0: Install and activate Visual Studio Enterprise 2022 on your own Windows laptop

Get a free copy of MS Visual Studio C++ through Microsoft's **Azure Dev Tools for Education** .

For instructions for UK students, start with this URL:
http://www.cs.uky.edu/docs/users/msdnaa.html#microsoft-azure-dev-tools

Follow the instructions, install and activate Visual Studio Enterprise 2022 on your own windows laptop. ***During the installation, do apply "Product Key" as described in the above instructions, otherwise the software license will be expired in 90 days.***

Part 1: Visual Studio

1. Log in to the lab computer and start Visual Studio (**Start → All Programs → Visual Studio 2022**).

   Note if you see the popped window asking you to sign in to use the Visual Studio 2022, type your link blue account followed by @uky.edu and password to sign in, password is not needed from the lab computer since you log in to the lab computer with your link blue account. Unless you choose to sign out before exiting Visual Studio 2022, you do not need to sign in to use the IDE next time)
2. **Create a New C++ project** (see the screenshot at **Figure 1**)
   a. **File → New → Project...** or **Ctrl+Shift+N**
   b. Select **Empty Project** then press **Next**
   c. Configure your new project (see **Figure 2**)
      1) Project Name: In the **"Project Name"** text field, enter a name for your project, for example, "Lab1".

2) Location: you may leave it in the …\repos\ folder, or use another folder where you will keep all CS215 labs, projects, etc. **Remember this folder! You will need it to submit to Canvas.**
3) Check the “Place solution and project in same directory” box.
4) Click the **Create** button in the bottom/right.

3. Now your project will appear on the side of the window named "Solution Explorer" (see **Figure 3**), with folders for "Header Files", “Source Files”, and so on.
4. **Add a new source file**:
   a. Right click on **Source Files** and select **Add → New Item...** (see **Figure 3**)
   b. Choose "C++ File (.cpp)".
   c. Enter a name for your file (for example, "Lab1.cpp" or "main.cpp" or leave it as the default "Source.cpp") and click **Add**. **Note: this is the file you will submit in Canvas.** Remember or make a note of the Location and File Name for future reference.

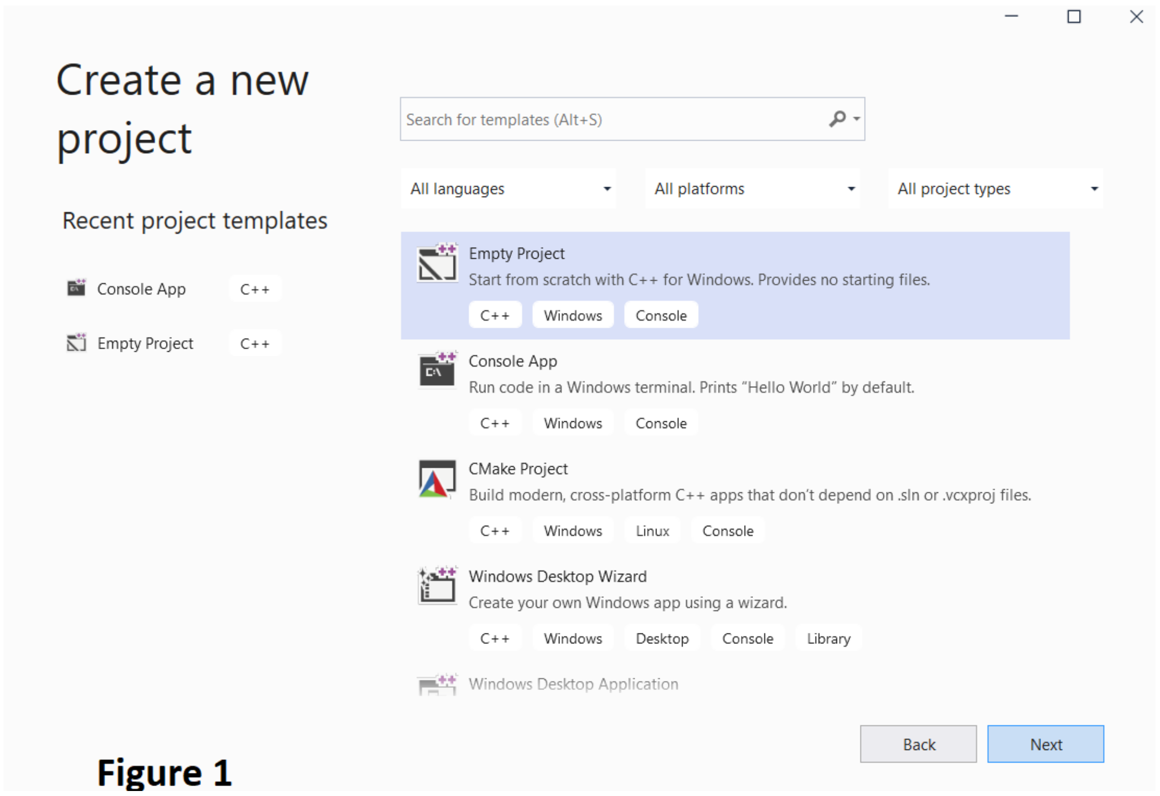
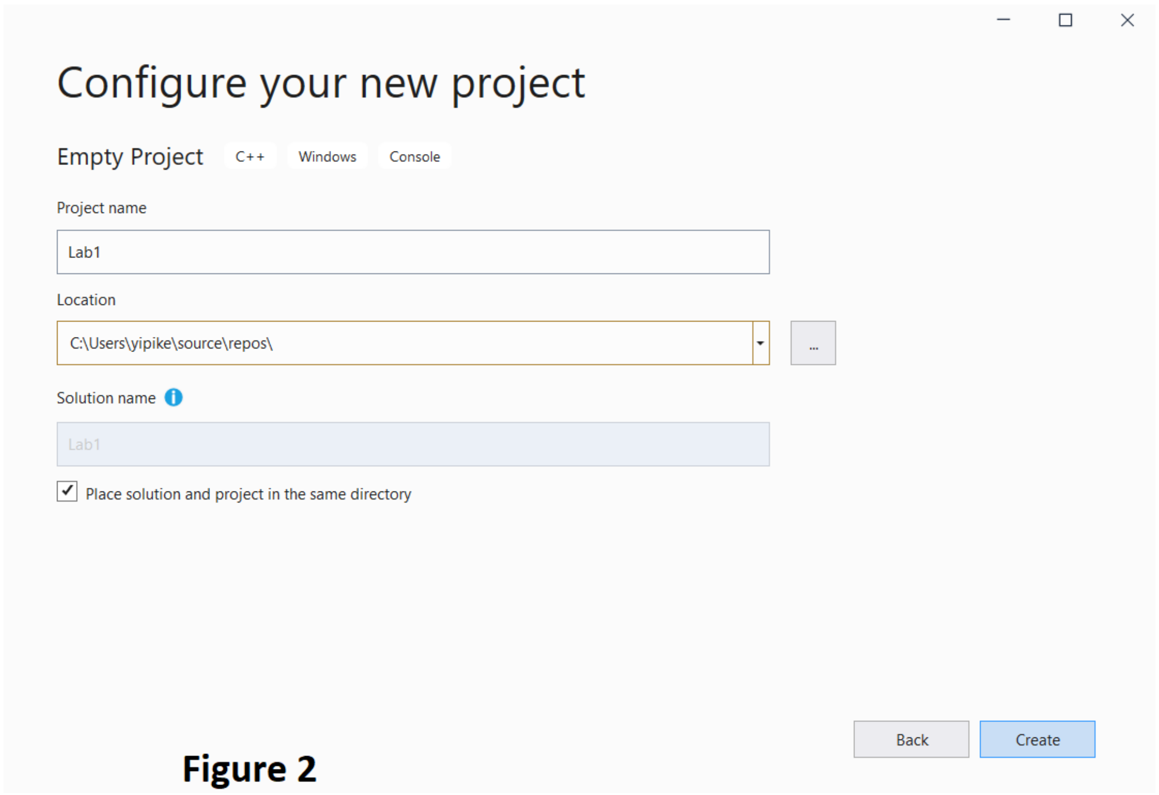

Create a new project
Search for templates (Alt+S)
All languages
All platforms
All project types
Recent project templates
Console App C++
Empty Project C++
Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console
Console App
Run code in a Windows terminal. Prints “Hello World” by default.
C++ Windows Console
CMake Project
Build modern, cross-platform C++ apps that don't depend on .sln or .vcxproj files.
C++ Windows Linux Console
Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library
Windows Desktop Application
Back
Next

Figure 1



Configure your new project

Empty Project C++ Windows Console

Project name

Lab1

Location

C:\Users\yipike\source\repos\

Solution name

Lab1

☑ Place solution and project in the same directory

Figure 2

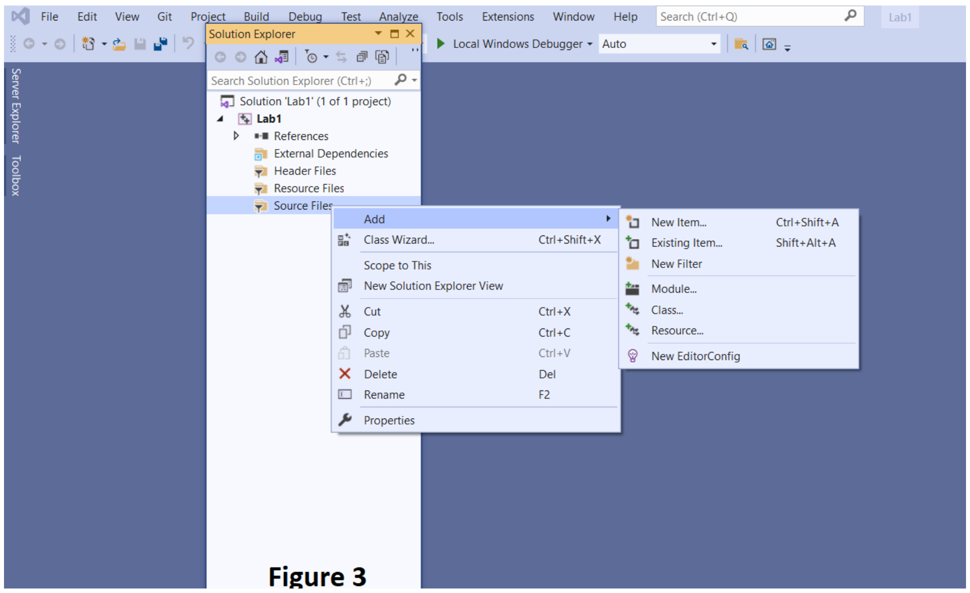
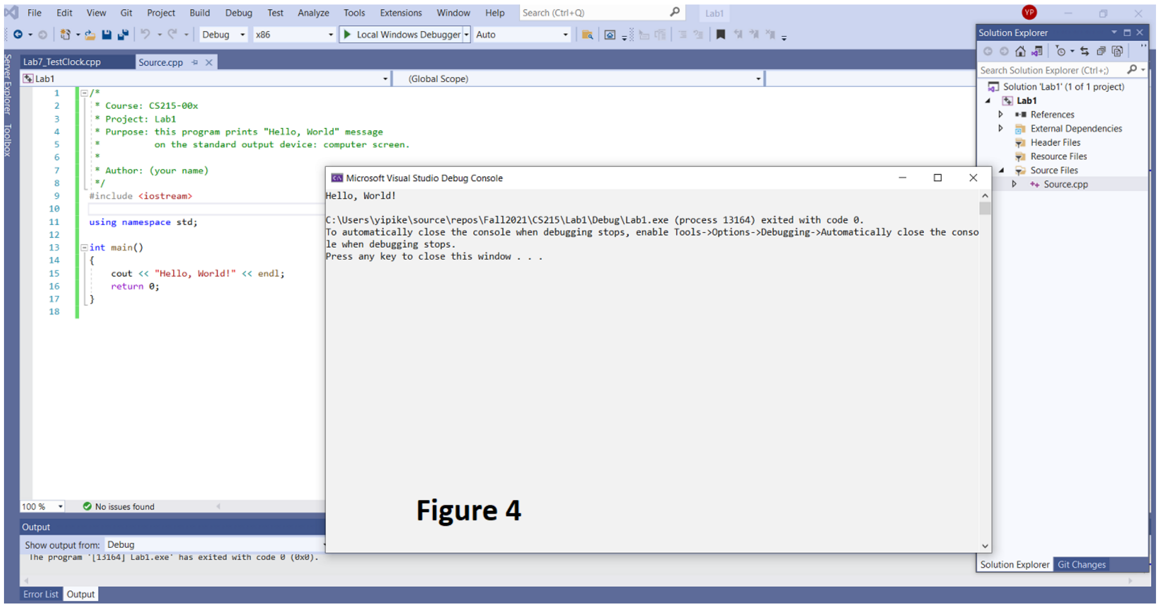


Figure 3

5. You will then by presented with an edit window. Enter your first C++ program to print “Hello, World!” to the computer screen and save.
6. Now let's try compiling:
   a. First, make sure you have saved your project (**Ctrl+Shift+S**).
   b. From the menu select **Build** → **Build Solution**, or press **Ctrl+Shift+B**. Check the "Output" tab at the bottom of the screen. It should say "1

succeeded, 0 failed". If not, check your program for errors, correct them, and try again.

7. Now run the program: from the menu select **Debug → Start Debugging**, or press **F5**. You will see the output of your program and ask you to press any key to close the window (shown in **Figure 4**).



```cpp
/*
 * Course: CS215-00x
 * Project: Lab1
 * Purpose: this program prints "Hello, World" message
 *          on the standard output device: computer screen.
 *
 * Author: (your name)
 */
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello, World!" << endl;
    return 0;
}
```

```
Hello, World!

C:\Users\yipike\source\repos\Fall2021\CS215\Lab1\Debug\Lab1.exe (process 13164) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

```
The program '[13164] Lab1.exe' has exited with code 0 (0x0).
```

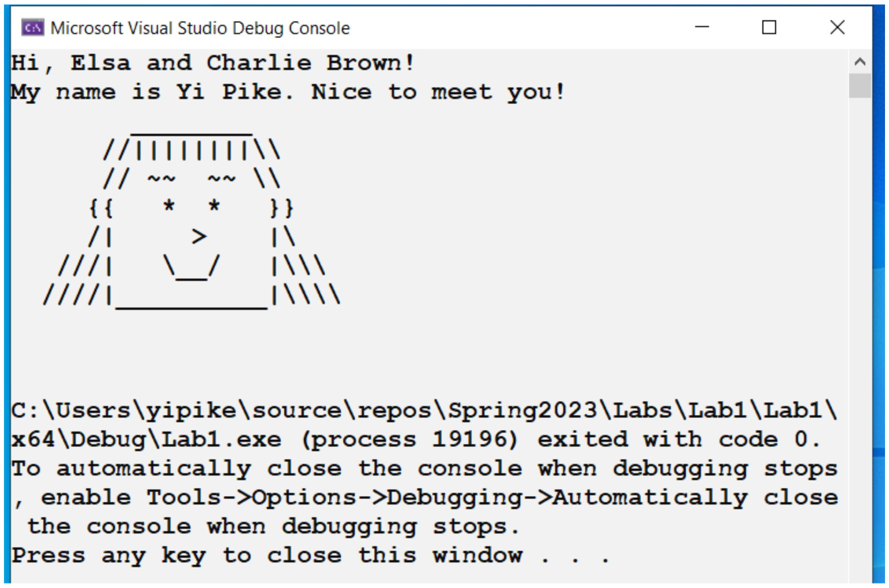
Figure 4

8. Go back to the edit window, and make some change to the .cpp file, so that your program should display two lines of message on the computer screen:
   Hello, A and B!
   My name is C. Nice to meet you!
   Please note that you need to find out your TWO Teaching Assistants' names and to replace items A and B from above example. Then use your name to replace item C. Remember to **save** your program.
9. You can try to run your program again by click "**Local Windows Debugger**" with a green arrow, then check if the required message has been displayed on the computer screen.

Part 2: Display your smiley face

1. Open the project you created from **Part 1**, modify the source file you created from **Part 1** so that your program will print your smiling face on the screen after displaying two lines of text message. (Note that :) or 😊cannot be used as a smiley face for this assignment)
2. Compile and run your program just as you do in **Part 1**.
3. Save your source file or simply press **Ctrl+S.**

A sample output is shown in **Figure 5**: (Assume your TAs are Elsa and Charlie Brown, and your name is Yi Pike.)



```
Hi, Elsa and Charlie Brown!
My name is Yi Pike. Nice to meet you!

        ________
      //||||||||\\
      // ~~  ~~ \\
     {{   *  *   }}
     /|     >    |\
   ///|   \__/   |\\\
  ////|__________|\\\\



C:\Users\yipike\source\repos\Spring2023\Labs\Lab1\Lab1\
x64\Debug\Lab1.exe (process 19196) exited with code 0.
To automatically close the console when debugging stops
, enable Tools->Options->Debugging->Automatically close
 the console when debugging stops.
Press any key to close this window . . .
```

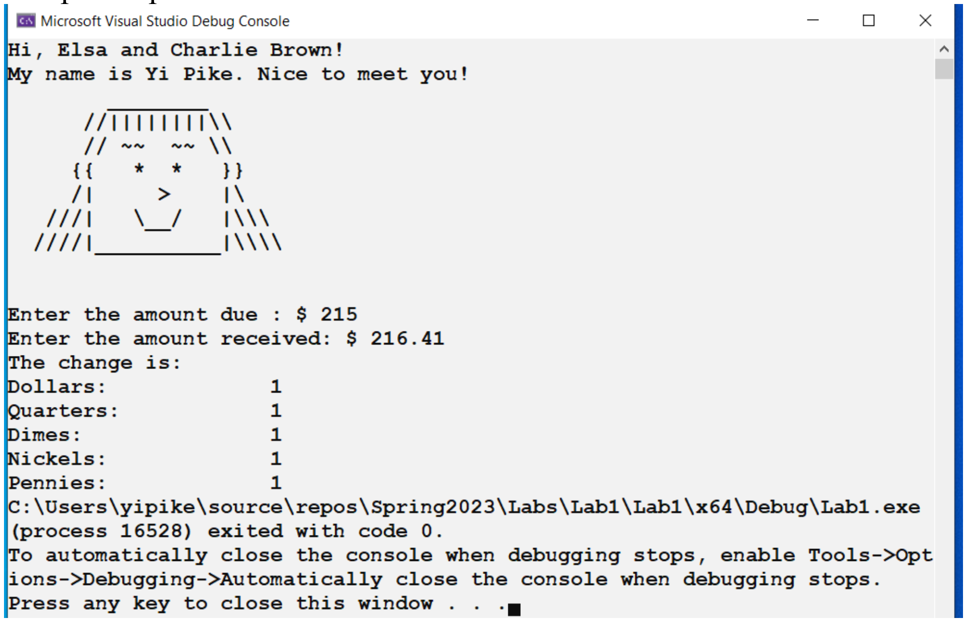
Figure 5

Note that you should use your own creative idea to print your own smiling face.

Part 3: Collect the user input and do a little calculation

1. Back to the edit window, and continue writing your program so that it can direct a cashier how to give change. This part of your program has two inputs: the amount due and the amount received from the customer (expressed in dollars). Display the dollars, dimes, nickels, and pennies that the customer should receive in return.
2. Compile and run your program just as you do in **Part 2**.
3. Save your source file or press **Ctrl+S.**
4. The following are sample outputs of running your program four times:

Sample output 1:

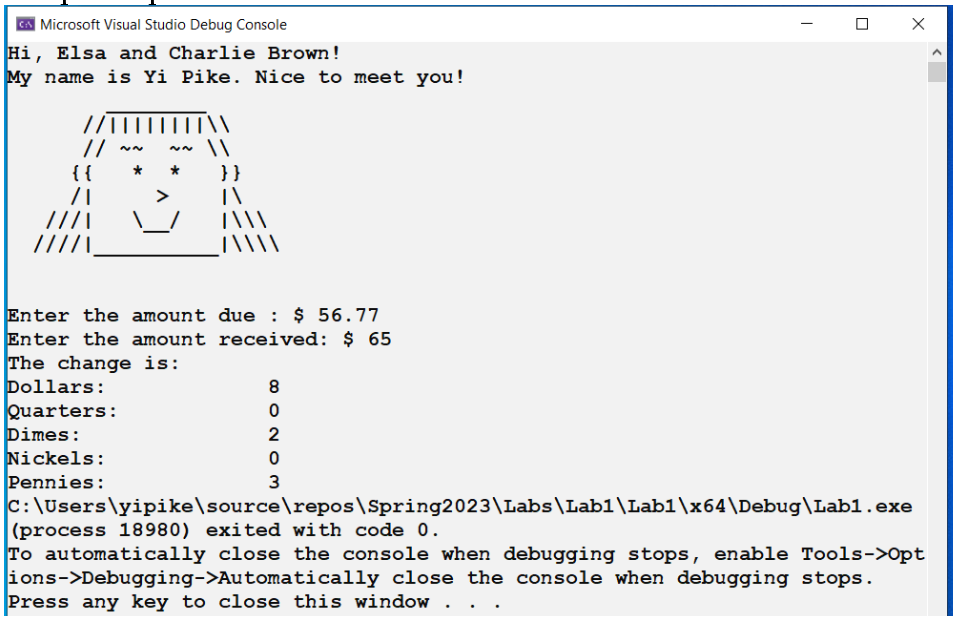


```
Hi, Elsa and Charlie Brown!
My name is Yi Pike. Nice to meet you!

        ________
      //||||||||\\
      // ~~  ~~ \\
     {{   *  *   }}
     /|     >    |\
   ///|   \__/   |\\\
  ////|__________|\\\\


Enter the amount due : $ 215
Enter the amount received: $ 216.41
The change is:
Dollars:            1
Quarters:           1
Dimes:              1
Nickels:            1
Pennies:            1
C:\Users\yipike\source\repos\Spring2023\Labs\Lab1\Lab1\x64\Debug\Lab1.exe
(process 16528) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Opt
ions->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

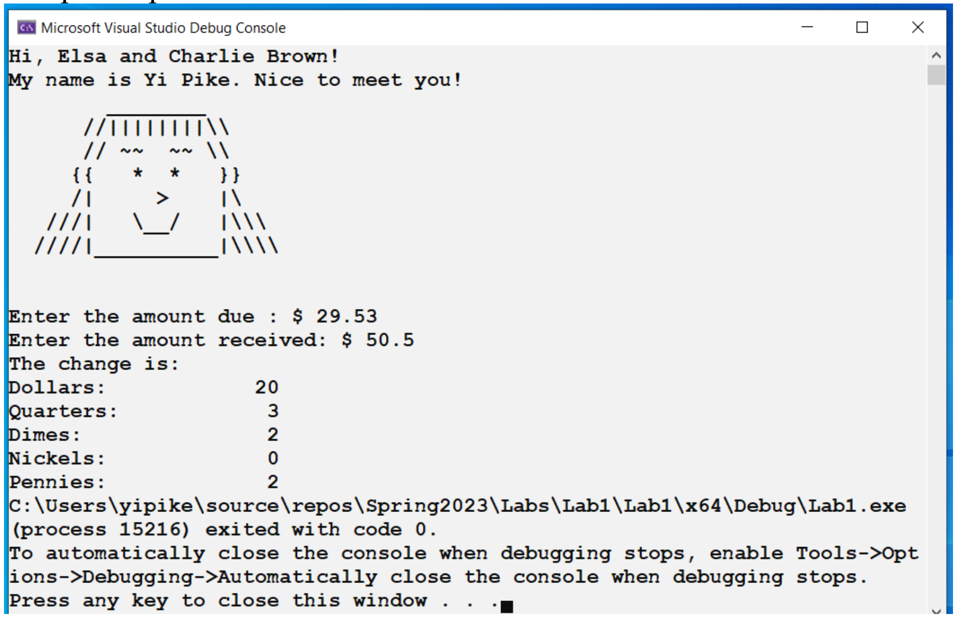
Sample output 2:



```
Hi, Elsa and Charlie Brown!
My name is Yi Pike. Nice to meet you!
        ________
      //||||||||\\
      // ~~  ~~ \\
     {{   *  *   }}
     /|     >    |\
   ///|   \__/   |\\\
  ////|__________|\\\\


Enter the amount due : $ 56.77
Enter the amount received: $ 65
The change is:
Dollars:             8
Quarters:            0
Dimes:               2
Nickels:             0
Pennies:             3
C:\Users\yipike\source\repos\Spring2023\Labs\Lab1\Lab1\x64\Debug\Lab1.exe
(process 18980) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Opt
ions->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

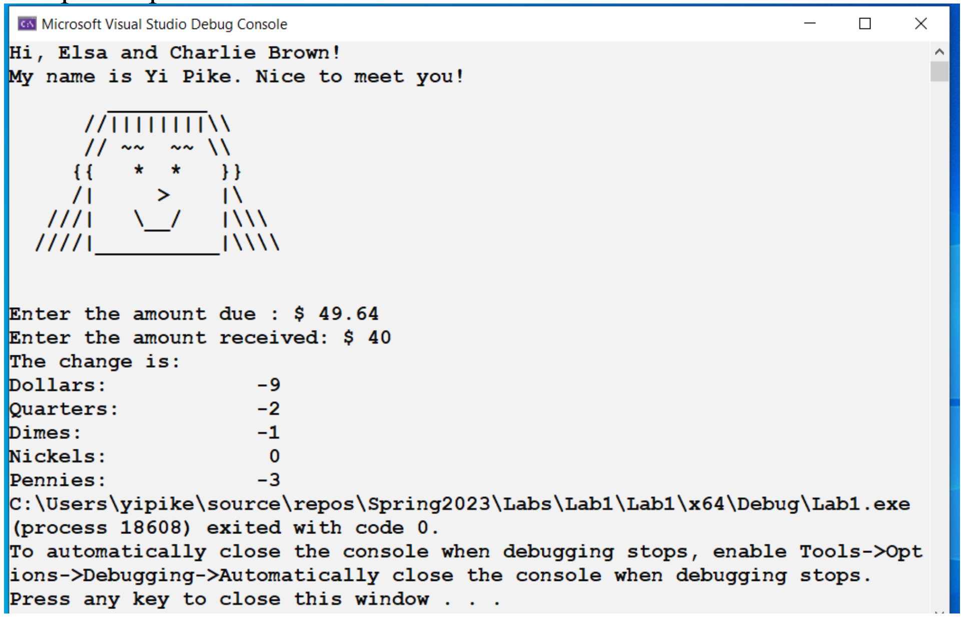
Sample output 3:



```
Hi, Elsa and Charlie Brown!
My name is Yi Pike. Nice to meet you!
        ________
      //||||||||\\
      // ~~  ~~ \\
     {{   *  *   }}
     /|     >    |\
   ///|   \__/   |\\\
  ////|__________|\\\\


Enter the amount due : $ 29.53
Enter the amount received: $ 50.5
The change is:
Dollars:            20
Quarters:            3
Dimes:               2
Nickels:             0
Pennies:             2
C:\Users\yipike\source\repos\Spring2023\Labs\Lab1\Lab1\x64\Debug\Lab1.exe
(process 15216) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Opt
ions->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Sample output 4:



```
Hi, Elsa and Charlie Brown!
My name is Yi Pike. Nice to meet you!

        ________
      //||||||||\\
      // ~~  ~~ \\
     {{   *  *   }}
     /|     >    |\
   ///|   \__/   |\\\
  ////|__________|\\\\


Enter the amount due : $ 49.64
Enter the amount received: $ 40
The change is:
Dollars:            -9
Quarters:           -2
Dimes:              -1
Nickels:             0
Pennies:            -3
C:\Users\yipike\source\repos\Spring2023\Labs\Lab1\Lab1\x64\Debug\Lab1.exe
(process 18608) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Opt
ions->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Note that Sample output 4 is the case, when the amount received is less than the amount due, which will generate the negative number in the change 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 received is not enough for the purchase. For now, you can leave it as it is shown in "Sample output 4". After learning the conditional statements in Chapter 3, we will do better by making decisions under different conditions.

To collect user input: please remember that users need 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 (and 2 points for your grade in this Lab Assignment☺)

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 in the change to return to the customers. If the number is not six characters, spaces are printed before the number.

Congratulations! You are ready to demonstrate your program to your TA.

Part 4: Demonstrate your code to your TA, then submit it in Canvas

1. Each Lab assignment needs to demonstrate to your TA to be graded. You can demonstrate Lab1 during Lab1 class (with possible bonus 3 points) or no later than the end of Lab2 class (this is the **demonstration deadline** for Lab1).
If you finish Lab1 assignment during Lab1 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 Lab2 class.

However, any demonstration later than the end of the Lab1 class cannot get bonus 3 points.)
If you need extra time, you can continue working on Lab1 assignment after the Lab class, and try to finish it before the next Lab class. Then demonstrate your Lab1 during Lab2 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 file through link "**`Lab 1`**". See Part 1 step 4 (highlighted in blue) to find which file to submit.
Even if you successfully demonstrated it to the TA, if you do not submit 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).
   - display your TAs' names correctly and your name correctly. (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. (4 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 have to 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!