# Data Types and Operators

**NOTE** - due to the problems that Visual Studio has with the U: network drive, it will be a good idea to develop and debug your programs on the desktop, or on an external drive or memory stick. In this case please remember to **keep an up to date copy on the U: drive** to be reviewed at any time. Ideally copy it across before the end of the session. If you are using a copy on the desktop be **very** careful that you **do not log off before copying your work back to the U: drive**, otherwise all your work will be **wiped and unrecoverable**. It’s also recommended to do a quick temporary copy to the U: drive a few times per session (maybe every 15-20 minutes, or after entering a large section of code), in case there is a power failure or your PC crashes.

Name the exercises **[your name] Tutorial xx Exercise yy**, with appropriate xx and yy.

## Exercise 01

1. Create a new Visual Studio project with a new source code file called main.cpp.
2. Add **comments** at the start to identify the program.
3. Add #include <conio.h> and #include <iostream> in the correct place
4. Add using namespace std; in the before your main() function
5. Add a main() function, don't forget the **braces** that form the statement block that will contain all of the statements.
6. Declare three integer variables, named appropriately to store user-entered values;
7. For each variable, display "Input a number" and use cin to get a number from the keyboard.
8. Once all three numbers have been entered, **display** them in reverse order to how they were entered, e.g. if 1, 3 and 5 are entered the program should display 5, 3, 1.
9. Don't forget to use \_getch() to pause the program before it finishes.
10. Use the debugger to step through the code to check for errors.
11. Edit the code to **display** all three numbers and their sum on the screen in the format: Number 1 + Number 2 + Number 3 = Sum
12. Calculate the **average** of the 3 numbers and display it on the next line (Evaluate the average **in place** if you can).

## Exercise 02

1. Create a new project and add main.cpp file, includes and main() function as usual.
2. Declare three **floating point** variables named appropriately.
3. Repeat the same steps for Exercise 1, noting any differences in the results. You should see some – why are there differences?
4. Use the debugger to step through the code to check for errors and understand how integers differ from floats.

## Exercise 03

1. Create a new project as usual.
2. Ask the user for a floating point number that is the radius of a circle
3. Use that number to calculate the circumference and area of a circle, and display both of these. Consider where you might use a constant in the calculations. Hint: PI never changes.
4. Run the program again, and enter 1000 for the radius. Write in your logbook what is output to the screen.

## Exercise 04

1. Create a new project as usual.
2. Ask the user for two integer and two floating point numbers.
3. Use **all** of the appropriate arithmetic operators with the two integers, (e.g. a+b, a-b, etc.) and display the results.
4. Use **all** of the appropriate arithmetic operators with the two floats, and display the results.
5. Use all possible arithmetic operators with one of the floats and one of the integers, and display the results. Write in your logbook and in comments what happens. Research and try to explain any unexpected results.
6. Now look at your code and consider where you could use the **compound** **assignment** operators. Wherever you can use them properly, notice how much smaller and more concise the code becomes.
7. Step through the code in the debugger to observe how these operators work and where the results of operations are different that you expected.

## Exercise 05

1. Create a new project as usual.
2. Ask the user to enter their first name and store that value in a string variable.
3. Ask the user to enter their last name and store that value in a string variable.
4. Display the user’s name in a few different format: first-name then last-name, last-name comma first-name, and last-name comma first initial. Is it easier to store the results of these different formats in temporary variables, or just use the variables you already declared?

## Exercise 06

1. Continue with the previous project
2. Now ask the user for three integers representing their birthday: date, month, and year.
3. Display the user’s information all together in a sensible and readable format. Do this by creating a temporary string variable, and then adding all of the other values to it. Pay attention to the need for spaces, endl and other punctuation to make it readable.
4. Step through the code in the debugger to figure out where any problems occur and make the output perfect.

## Exercise 07

1. Create a new project as usual and #include the necessary files.
2. Ask the user to enter a current temperature in **Celsius** and convert it to **Fahrenheit**, using the equation:
3. Use best practices for variable names and such, displaying the proper messaging and resulting value.
4. Don’t forget to add \_getch() so the message can be read before the problem exits.
5. Implement using both integer and float and compare the difference in values. You can also test for loss of data by converting back to Celsius to see if you get the starting number.

## Exercise 08

1. Create a new project as usual and #include the necessary files.
2. Write the few lines of code necessary to generate a random number and store it in an integer variable. Refer to the MicroLesson on **Random Numbers** if you're not sure how to do this.
3. Use the modulus operator to reduce the random number to a specific range, like 1 to 10.
4. Ask the user to guess what the number is. (Maybe consider using a constant string for the instructions?) Display a message after the guess telling the user if they are right or wrong.
5. Don’t forget to add \_getch() so the message can be read before the problem exits.
6. Modify the program so the final message gives a hint based on how far off the guess is.

## Challenge

1. Create a project with a *separate* function that simulates a dice roll. Dice come all kinds of sizes – 4-sided, 6-sided, 20-sided… What makes the most sense?
2. Call this function from main()and display the results of the roll.
3. How would you modify the code to roll multiple dice? Or to combine dice with different numbers of sides?

**REMEMBER** - don't forget to copy the finished project to the U: drive, especially if the project was on the desktop.