COMSC 260

Fall 2020

Programming Assignment 3

Worth 15 points (1.5% of your grade)

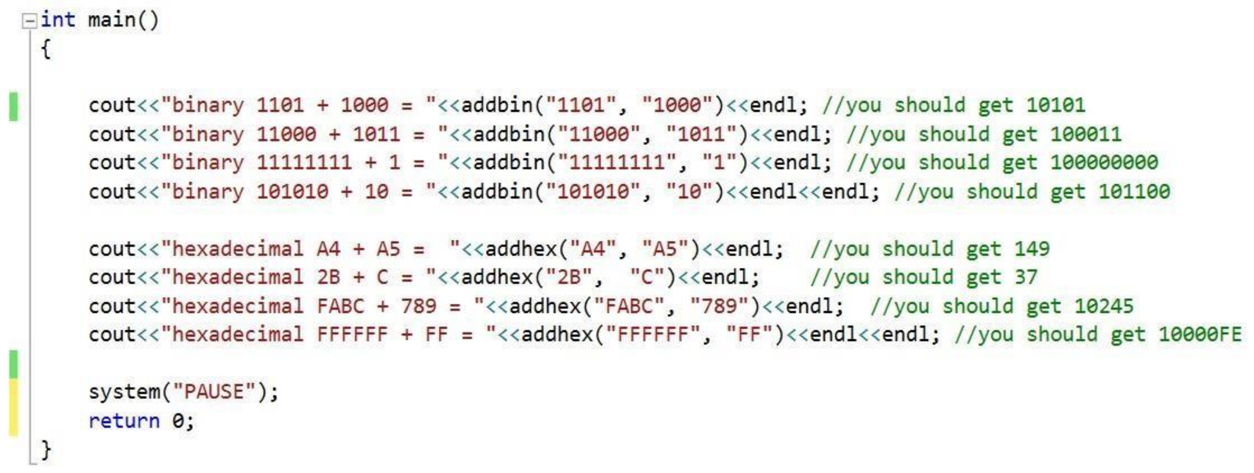
DUE: Tuesday, 9/29/20 by 11:59 P.M. on Canvas

Start by downloading the **260_assign3.cpp** file from the Programming Assignment 3 folder on Canvas

NOTE: Your submission for this assignment should be a single **.cpp** file and a single **.pdf** file. The following naming convention should be used for naming your files: **firstname_lastname_260_assign3.cpp and firstname_lastname_260_assign3.pdf**. The pdf file that you submit should contain the screenshots of your sample runs of the program (see below). For example, if your first name is "James" and your last name is "Smith", then your files should be named James_Smith_260_assign3.cpp James_Smith_260_assign3.pdf.

COMMENTS (worth 7.5% of your programming assignment grade): Your program should have at least **ten (10)** different detailed comments explaining the different parts of your program. Each individual comment should be, at a minimum, a short sentence explaining a particular part of your code. You should make each comment as detailed as necessary to fully explain your code. You should also number each of your comments (i.e., comment 1, comment 2, etc.). **Important Note: Any of my comments in the program that you download from Canvas do NOT count towards the ten comments!**

SAMPLE RUNS (worth 7.5% of your programming assignment grade): You should submit screenshots of at least **five (5)** different sample runs of your program. Each sample run needs to use different inputs, and your sample runs should **NOT** be the same as the sample runs that are used in this write-up for the assignment. You should also number each of your sample runs (i.e., sample run 1, sample run 2, etc.). All of your sample runs should follow this format – for each individual sample run, screenshot the code with your function calls in main **AND** the corresponding output on the console screen:

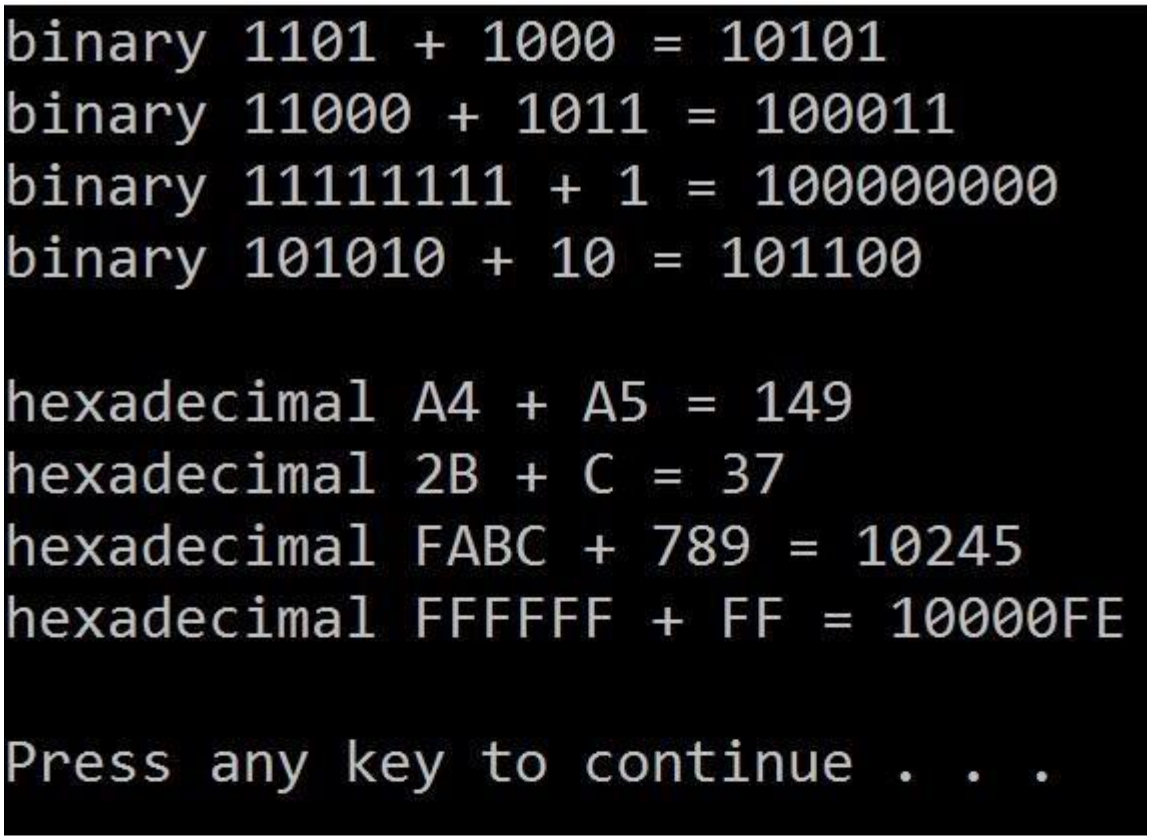


```
int main()
{

    cout<<"binary 1101 + 1000 = "<<addbin("1101", "1000")<<endl; //you should get 10101
    cout<<"binary 11000 + 1011 = "<<addbin("11000", "1011")<<endl; //you should get 100011
    cout<<"binary 11111111 + 1 = "<<addbin("11111111", "1")<<endl; //you should get 100000000
    cout<<"binary 101010 + 10 = "<<addbin("101010", "10")<<endl<<endl; //you should get 101100

    cout<<"hexadecimal A4 + A5 =  "<<addhex("A4", "A5")<<endl;  //you should get 149
    cout<<"hexadecimal 2B + C = "<<addhex("2B",  "C")<<endl;    //you should get 37
    cout<<"hexadecimal FABC + 789 = "<<addhex("FABC", "789")<<endl;  //you should get 10245
    cout<<"hexadecimal FFFFFF + FF = "<<addhex("FFFFFF", "FF")<<endl<<endl; //you should get 10000FE

    system("PAUSE");
    return 0;
}
```

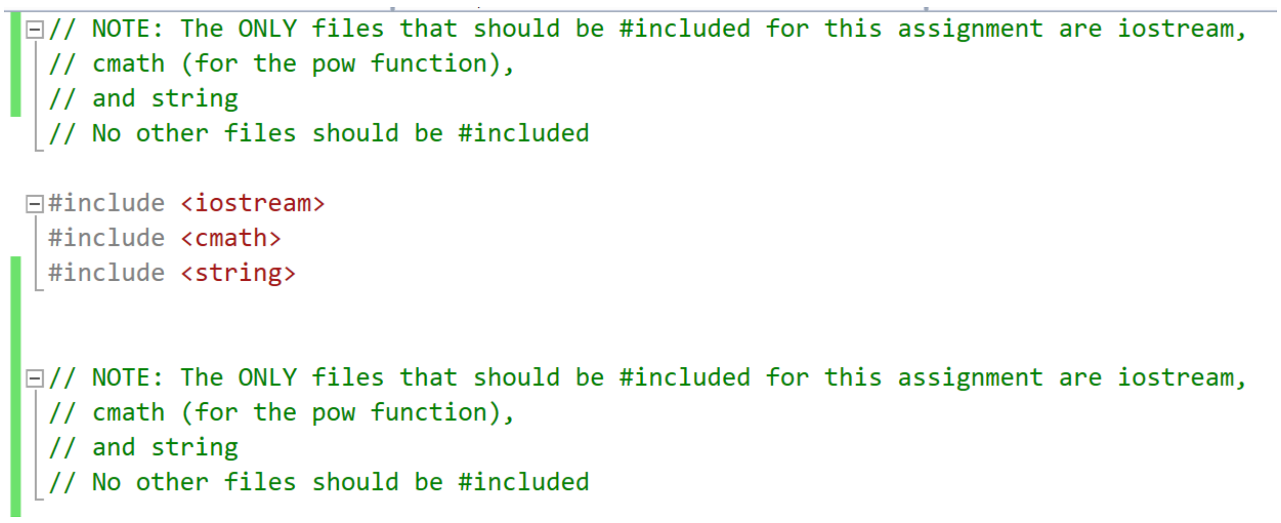


```
binary 1101 + 1000 = 10101
binary 11000 + 1011 = 100011
binary 11111111 + 1 = 100000000
binary 101010 + 10 = 101100

hexadecimal A4 + A5 = 149
hexadecimal 2B + C = 37
hexadecimal FABC + 789 = 10245
hexadecimal FFFFFF + FF = 10000FE

Press any key to continue . . .
```

NOTE: The **ONLY** files that should be #included for this assignment are **iostream, cmath (for the pow function),** and **string**. All three of these files are already #included for you in the 260_assign2.cpp file that you will be downloading from Canvas. **No** other files should be #included for this assignment.



```
// NOTE: The ONLY files that should be #included for this assignment are iostream,
// cmath (for the pow function),
// and string
// No other files should be #included

#include <iostream>
#include <cmath>
#include <string>


// NOTE: The ONLY files that should be #included for this assignment are iostream,
// cmath (for the pow function),
// and string
// No other files should be #included
```

For your third programming assignment you will be implementing the following two functions in a C++ program:

string addbin(string, string);

// adds two binary numbers together and returns the sum

string addhex(string, string);

// adds two hexadecimal numbers together and returns the sum

NOTE: In some cases, you may need to **sign-extend** a number when doing addition. For example, when you add two binary/hexadecimal numbers, the two numbers need to be the same length. So if you are trying to do the addition:

A1FC + B

You need to actually make it A1FC + **000**B

After doing that you can add the numbers **column-by-column, starting at the right end (i.e., C + B first)**.

You need to have a variable to maintain the carry. Each time you add the numbers in a column, it is possible that a carry will be generated, which means a 1 is carried into the next column to the left. The carry variable should always have a value of 0 or 1.

Treat the numbers as **UN**signed numbers, so you can sign-extend a binary/hexadecimal number by simply adding the appropriate number of **0**'s to the left.

As another example, the addition 1 + ABCD would need to be **000**1 + ABCD

Since binary/hexadecimal numbers are represented as strings, this means you need to **concatenate** the **character '0'** with the string (possibly several times) to do the sign-extension.

Also, if there is a carry out of the most significant bit (MSB), make sure to include that in your answer; otherwise it will be incorrect. So the sum string can be larger than the two input strings. For example, binary 1 + 1 = 10, because 1+1=0, and you carry 1 over to the next column.

Also remember the formula we learned in class to recalculate the sum when the sum is larger than the base of the number system:

sum = sum mod base

sum = sum mod 2 (for binary)

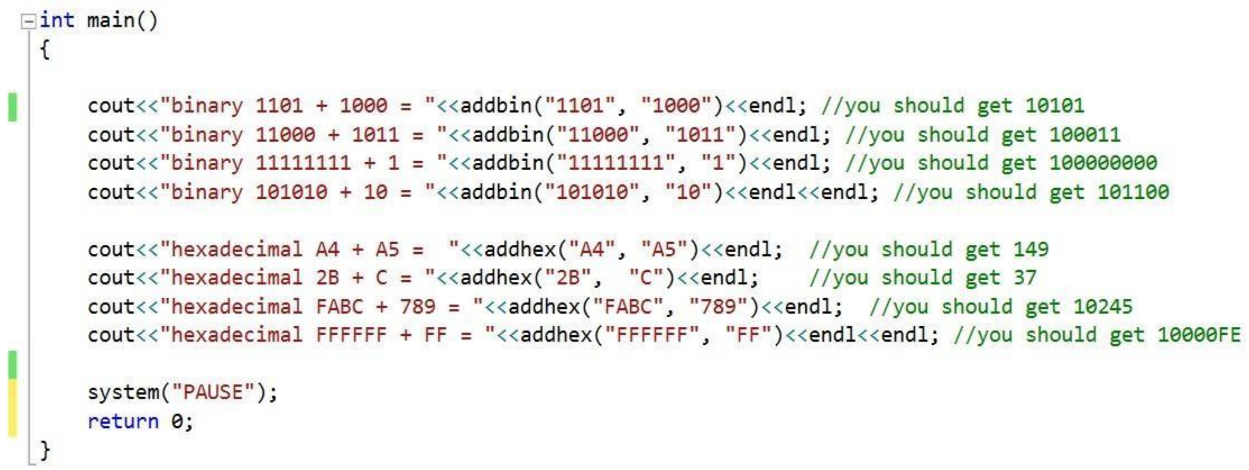
sum = sum mod 16 (for hexadecimal)

Do you see why the formula works for both numbering systems? Make sure you are comfortable with it as it will appear in quizzes and tests.

NOTE: The numbers need to be added **directly** in their hexadecimal or binary format. For example, it is **NOT** allowed to convert the numbers to decimal first, add the decimal values together, and then convert the result back to binary or hexadecimal. Your functions for adding binary and hexadecimal numbers should work **exactly** the same as the algorithm we used in class for adding binary and hexadecimal numbers together by hand.

Sample Run:

Here is a sample run of the program. After you finish implementing the two functions, run the program with these test cases (which I've provided to you in the 260_assign3.cpp file) and make sure you get the same results. The test cases also show you how to properly call each function. Feel free to come up with and run your own test cases as well.

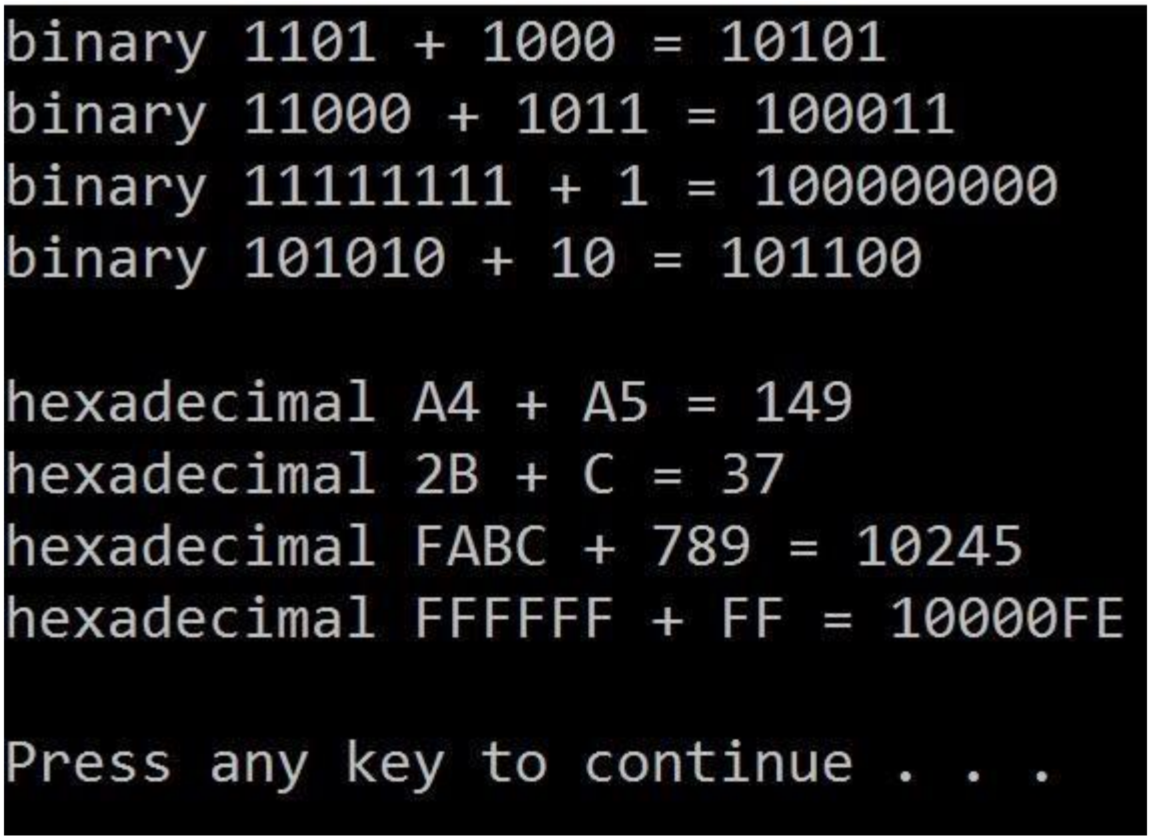


```
int main()
{

    cout<<"binary 1101 + 1000 = "<<addbin("1101", "1000")<<endl; //you should get 10101
    cout<<"binary 11000 + 1011 = "<<addbin("11000", "1011")<<endl; //you should get 100011
    cout<<"binary 11111111 + 1 = "<<addbin("11111111", "1")<<endl; //you should get 100000000
    cout<<"binary 101010 + 10 = "<<addbin("101010", "10")<<endl<<endl; //you should get 101100

    cout<<"hexadecimal A4 + A5 =  "<<addhex("A4", "A5")<<endl;  //you should get 149
    cout<<"hexadecimal 2B + C = "<<addhex("2B",  "C")<<endl;    //you should get 37
    cout<<"hexadecimal FABC + 789 = "<<addhex("FABC", "789")<<endl;  //you should get 10245
    cout<<"hexadecimal FFFFFF + FF = "<<addhex("FFFFFF", "FF")<<endl<<endl; //you should get 10000FE

    system("PAUSE");
    return 0;
}
```



```
binary 1101 + 1000 = 10101
binary 11000 + 1011 = 100011
binary 11111111 + 1 = 100000000
binary 101010 + 10 = 101100

hexadecimal A4 + A5 = 149
hexadecimal 2B + C = 37
hexadecimal FABC + 789 = 10245
hexadecimal FFFFFF + FF = 10000FE

Press any key to continue . . .
```