Integrity Checking Using CRC

For this assignment you will write a CRC integrity checking program will have two modes of operation.  Your program must implement the CRC calculation method involving long division by a polynomial that we discussed in lecture. In "calculation" mode, the program will calculate the CRC-12 value for a given input string. In "verification" mode, the program will interpret the last 3 hex characters of an input string as a CRC-12 value, and it will then determine whether this is the correct CRC-12 value for the input string without those 3 characters. In both modes, the program must report header information, intermediate results, and final results as described further below. The mode and the input string to process will be supplied to your program as command line arguments. The program will be graded according to the Grading Rubric that appears at the bottom of this assignment.

Use the CRC polynomial  x12 + x11 + x8 + x7 + x5 + x4 + x2 + 1.

**Programming Language**

The program must be written in C, C++, or Java, whichever you find more convenient. No other programming or scripting languages are permitted. If you are coding in C or C++, you must use only the standard libraries, such as stdio.h, math.h, and the Standard Template Library. The reason for this is so that the graders will be able to compile and run your program on their machines.

If you are using Java, you must use only the classes and packages included in a standard "SE" edition of Java, but you may not use the BigInteger class. ***If your program is written in Java and it uses the BigInteger class, you will receive a grade of zero (0) for this assignment.***

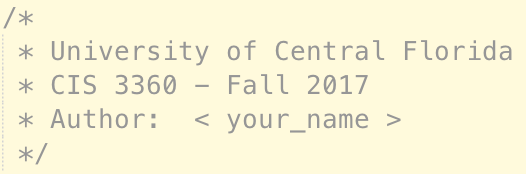
**What You Should Submit**

You should submit a single source code file (.c, .cpp, or .java)  for one of the permitted languages. Multiple submissions are permitted, but only the last submission before the deadline will be graded. Do not worry about the number suffix that Webcourses adds to the second and subsequent submissions -- the graders know how to handle it.  Now, if there are no submissions before the deadline, then only the first submission after the deadline will be graded with the point penalty described in the syllabus.

Your entire program should be contained in exactly one source code file, which should contain all classes, functions, and methods necessary to make your program run. C/C++ programmers should not use separate header files. Java programmers should not use a package statement.  This is so our test scripts can run without changes.

If you submit a C/C++ program, the suggested file name is "crcintegrity.c" or "crcintegrity.cpp". If your program is written in Java, the file (and hence the main class) must be named "CrcIntegrity.java".

Your program source file should have a comment header at the top identifying you as the program author. The header should use the following form. If you are teaming, both names should appear in the header.



**Please note:** we will not accept compiled versions of your program, nor will we accept multi-file programs. You must submit exactly one file, which must be a source code file. However, you may submit as many updated versions of your program file as desired, up to the submission deadline.

**Teaming**

You may develop by yourself or in teams of two students currently in the course. Teams of more than two students are not permitted. If you choose to work as a team:

1. Both students must submit the ***same*** file on Webcourses. If you do not personally submit a program file, you will get a grade of zero for the assignment. If you are working as a team and the files submitted are different, then both team members will receive 20-point deductions.
2. The source file that you submit must contain the required course header. There will be a 20-point deduction if your source file does not contain the required course header.
3. The course header in the source file that you submit must name ***both*** team members as the authors. If only one of you submits a header identifying both team members, then whoever forgot to name the other team member will receive a deduction of 20 points.

**Command Line Arguments**

The program must read in two command line arguments. If you are unsure what is meant by this, or how to use command line arguments, please review the article on this topic in the “Programming Resources” section of this Webcourse.

The programming resources article contains complete programs in C and Java that illustrate how to input and read command line arguments. If you are unfamiliar with using command line arguments, you are strongly advised to key in the appropriate sample program and to make it work on your system before proceeding with the program development for this assignment. Once you have mastered the sample program, you can then proceed to develop a separate program for this assignment. For this assignment, your program will only need two arguments.

Please note: Most IDEs, like Eclipse and NetBeans, require you to configure your program's project to pass a set of command line arguments to the program when it is run. You may wish to use the inputs from the sample outputs included in this assignment for development purposes, which illustrate both modes of operation.

Of course, setting up your IDE in this manner just configures it for just one particular set of command line arguments. Once your program works with this one set of arguments, it is more convenient to copy the source code into a new folder on your desktop where you will be able to use different files and parameters by simply typing them in on the command line and pressing the "Enter" button. Your program may NOT prompt the user to enter the arguments, nor wait for the user to enter the arguments, nor may it assume that they will have any particular names or values.

**The command line arguments for this program are as follows:**

1. The first argument will be a flag value that identifies the mode of operation: “c” for calculation mode, or “v” for verification mode. This value will always be provided to you in lower case, so you do not need to worry about checking whether they are upper case and then converting to lower case.  Please note that the actual argument will not contain quotation marks. Only these two values are allowed. Any other values should produce a simple error message and a graceful exit from the program.

2. The second argument will be the input string to process according to the mode. The string will always be a valid sequence of uppercase hexadecimal characters, so you do not need to validate this value or convert it to upper case. There may be as few as 3 or as many as 40 characters in this string. In verification mode, the last 3 hex characters will represent the "observed" CRC value, which you must check for validity.

**Compiling and Running from the Command Line**

Your program must compile and run from the command line because that is how we must test it. If you are unsure what is meant by this, please review the article on this topic in the “Programming Resources” section this Webcourse.

We will compile your program using one the following commands:

C program: prompt> gcc -lm -o CrcIntegrity crcintegrity.c

C++ program: prompt> g++ -lm -o CrcIntegrity crcintegrity.cpp

Java program: prompt> javac CrcIntegrity.java

Once the program is compiled, we will use a script to test your program against several different combinations of input arguments. Each program test configuration will be launched with command line parameters in the following form:

C/C++ program in Windows: prompt> CrcIntegrity [mode] [string\_to\_process]

C/C++ program in Linux: prompt> ./CrcIntegrity [mode] [string\_to\_process]

Java program on all systems: prompt> java CrcIntegrity [mode] [string\_to\_process]

Please note: the "[" and "]" brackets in the above command illustrations are for display purposes only. An actual command would look like: "CrcIntegrity c D2B" with no brackets or quotation marks.

**Testing Your Program**

You are strongly advised to test your program in the same manner that we will use to grade it. This means: (a) you should be able to compile and run it from the command line within a Command or Terminal window, and (b) you should use command line arguments. A good test will be to use the input arguments shown in the sample outputs below, so that you can compare your output to what is shown below.

If you are writing your program in C/C++ and you do not currently have the ability to compile and run your program from within a Command Window, you may wish to review the "Programming Resources" article on installing MinGW for Windows, which will give you access to the free gcc and g++ compilers for compiling and running C/C++ programs.

**Calculating CRC Values**

You will need to calculate CRC values in both modes of operation. In calculate mode, the value calculated is the final output. In verify mode, you must take one additional step, which is to compare the CRC value that you calculate against the value that is observed from the input string to determine whether the observed value is correct.

You should use the CRC calculation method we discussed in lecture, since this will produce the intermediate result strings that your program must output. Moreover, you should use arrays of characters, strings, or boolean values to hold the polynomial and the input string, since the input string can be as long as 40 hex characters (160 bits) and you will need to add 12 additional zero values as padding to start off the computation.  You cannot use a long integer value for this program since 40 hex characters represent 160 bits and a long integer is only 64 bits.

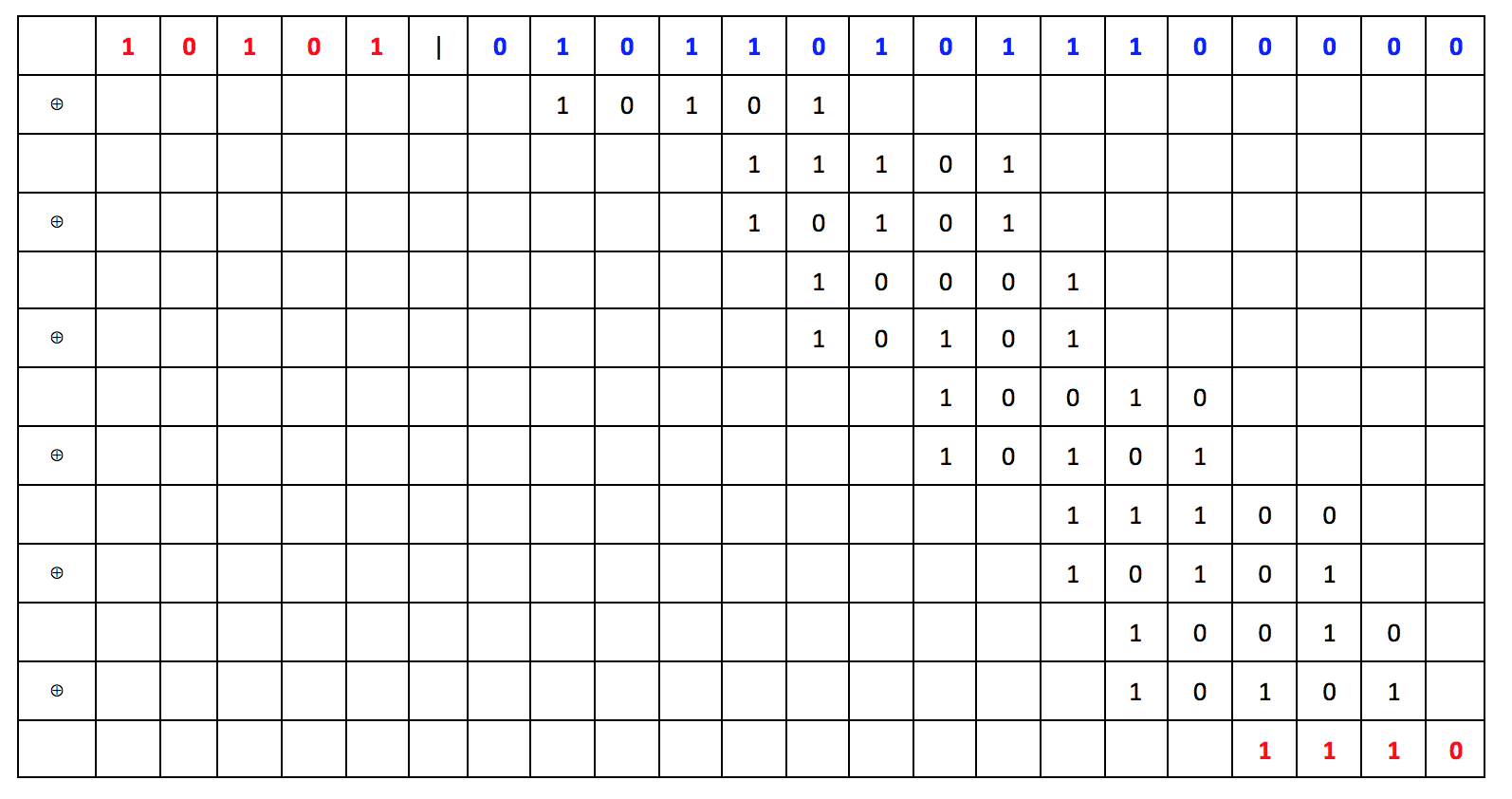
*Java programmers are warned (here, for the second time; see Programming Language section above) not to use Java's BigInteger class -- you will receive a zero for this assignment if you do. You should use arrays of characters, strings, or boolean values and perform the calculations as we have described them in lecture.*

The method discussed in lecture is illustrated using the following example in which we will compute the CRC-4 code for the message 5AE, using the polynomial x4 + x2 + 1. This is only a simple example, to illustrate the procedure. For the program that you must write, you will need to calculate CRC-12 values a well as perform verification as described.

To begin, we convert the polynomial for this example to the 5-bit string 10101. Next, we convert the input string 5AE to its binary equivalent, 010110101110. Next, we pad input value with 4 zeroes at the right since we are performing CRC-4 in this example, producing the value 0101101011100000, which is called the **dividend**. This is the value that we will divide by the polynomial. The remainder that we get from this division will be the desired CRC-4 value.

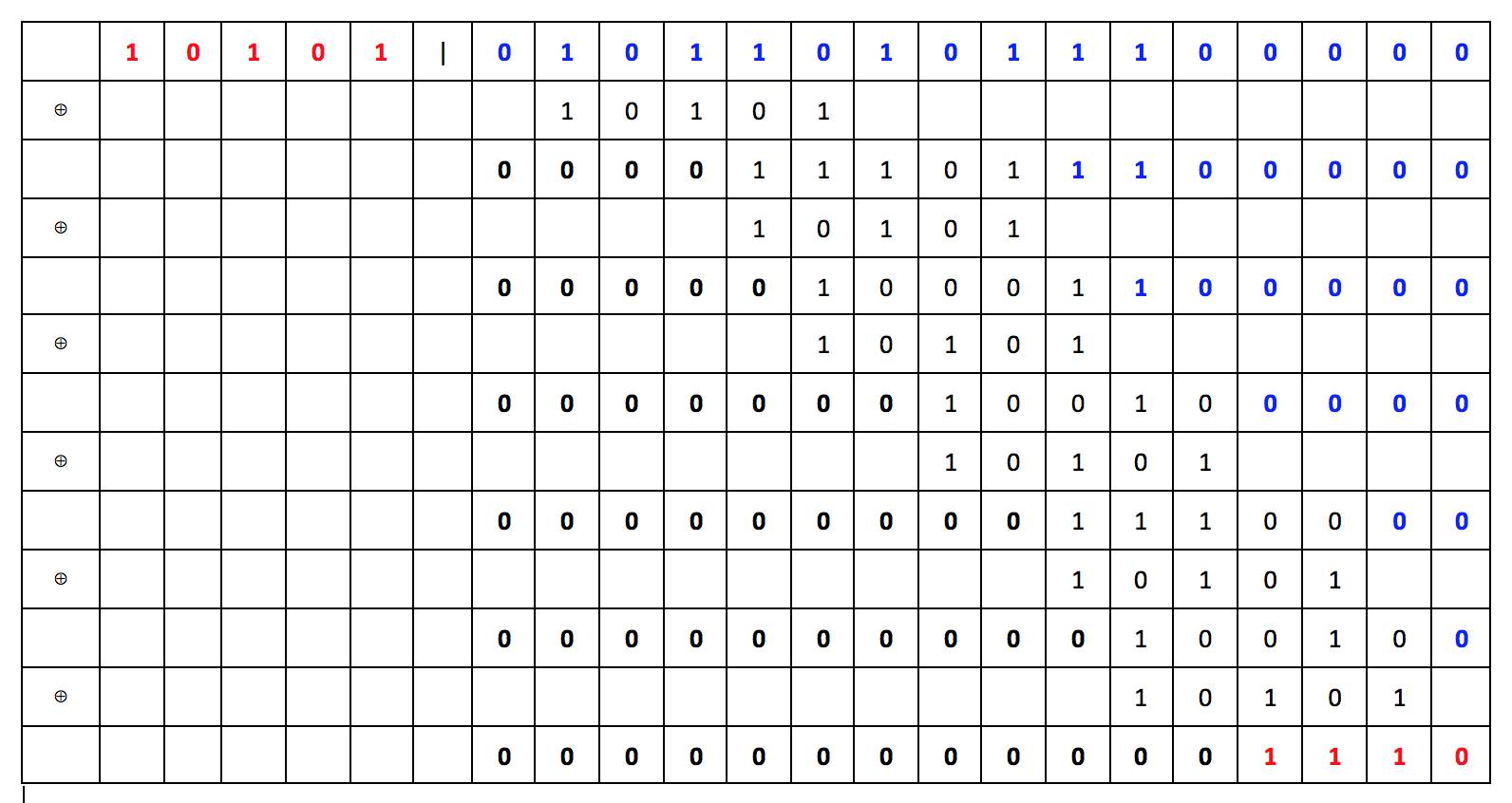
You will recall that to perform the binary division, we use the XOR operation for the subtraction part. We apply the XOR operator separately on each bit position involved in the subtraction. Other than the use of XOR, the division is performed in exactly the same way as long division for base 10 numbers.

The following table illustrates the division method we discussed in lecture. In the table, the dividend (padded input string) is shown in blue, the polynomial divisor is shown in red, and the results of the XOR operations are shown on the lines that do not have the XOR symbol at the left edge. The CRC value is the remainder, 1110, shown in red at the bottom. We show this binary value and express it equivalently as the hex character E.



Your program will need to output the results of each XOR operation in a format that contains the same number of bits for every result. The number of bits we wish to use in this example is 16, which is computed from the length of the binary input (3 hex characters = 12 bits), plus the number of pad characters (here, 4). So, we need to report the results of the XOR operations as 16-bit binary numbers. The way we will do this is to add leading zeroes to the left of each XOR result and to bring down the unused bits from the dividend to the right.

This is illustrated in the following table, which shows the same division calculation as above, only with the desired 16-bit output lines shown in bold. Please note that the first output value is simply the dividend (which includes the padding). The polynomial divisor is reported separately and is not included in the table of XOR operation results.



**Programming Tips**

One very convenient way to represent the binary numbers for these calculations is to represent them as strings and to perform all of the functions listed below on string values. This way, you will not need to worry about arithmetic overflow. As previously stated, our longer test cases will not be able to be expressed as long integers.

In developing your programs, you may also find it useful to write separate functions or methods for CRC calculation and CRC verification, and also for the following components:

* converting a hexadecimal string into binary string
* converting a binary string to hexadecimal
* an XOR function/method that takes two binary strings as input and returns the XOR result
* producing the header output
* producing the intermediate results output
* producing the final results output

Of course, some of the listed functions will very likely invoke others of these functions. It is good practice to reuse components instead of writing the same block of code in multiple places. Many typos and logical errors can be avoided by doing so, and if any corrections are needed, they can be done conveniently in just one place.

Remember: the basic idea is to break down the overall task into small pieces that can be developed (and tested) separately so you can build your program incrementally.

**Program Output**

The program must present output consisting of header information, intermediate results, and final results, as described separately below. Sample outputs follow in the next section.

***Header Information***

The header information must be written on separate lines as shown in the sample outputs that follow. The individual lines are:

1. The program must first write a string of 62 hyphens (dashes) to serve as a separator, and then, 2 lines below it, the string "CIS3360 Fall 2017 Integrity Checking Using CRC", and then on the next line the string "Author: " followed by your name (or names, if teaming) and a newline.

2. The program must then write the string "The input string (hex): ", followed by the string value of the second command line input argument

3. The program must then write the string "The input string (bin): ", followed by the binary value of the second command line input argument, and then a newline.

4. The program must then write the string "The polynomial used (binary bit string): ", followed by the 13-bit binary value of the polynomial given at the top of this assignment with spaces every 4 characters to improve readability.

5. The program must then write the string "Mode of operation: ", followed by either the word "calculation" or "verification", depending on the value of the first command line input argument

6. The content of this line depends on whether the mode is calculate or verify.

a. If in calculation mode, this line should read: "Number of zeroes that will be appended to the binary input: 12"

b. If in verification mode, this line should read: "The CRC observed at the end of the input: ", followed by the binary and hex values for the last 3 hex characters of the input string, as shown in the sample outputs that follow.

***Intermediate Results***

This section will begin with the string: "The binary string difference after each XOR step of the CRC calculation", followed by a newline.

Following the above statement, this section will show on separate lines the results of each XOR step as described above, with spaces every 4 characters as shown to improve readability.

Please note that for calculation mode, you must pad the binary version of the input string with 12 zeroes before you start the division process, but for verification mode you have two choices. The first choice is to remove the last three hex characters (i.e., the last 12 bits) representing the observed CRC, in which case you must then pad the rest with 12 zeroes as if you were simply calculating the CRC. The second choice is to leave the observed CRC attached to the rest of the input, in which case you should not add zeroes for padding. The difference between these two choices determines how you will interpret the result in the bottom row of the intermediate results output.

***Final Results***

The contents of the final results output section will depend on the mode.

For calculation mode, the final results section will consist of one line containing the string "The CRC computed from the input: ", followed by the binary and hex versions of the last 12 bits of the last intermediate output line, as shown in the sample output for calculate mode below.

For verification mode, the final results section will consist of two lines. The first line will report the computed CRC in binary and hex form in the same format as for calculation mode. The second line will consist of the string "Did the CRC check pass? (Yes or No): ", followed by "Yes" or "No", as appropriate.

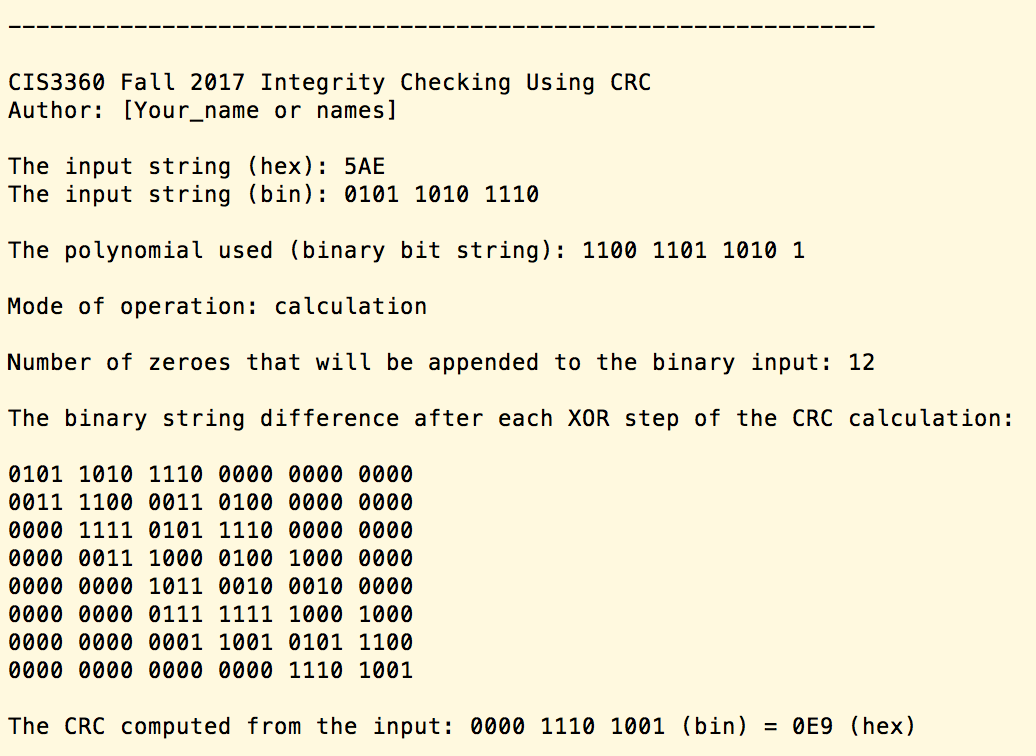
Please note:  How you determine the calculated CRC in verify mode depends on whether you include or strip the observed CRC from or included dividend, as described in the previous section. If you strip the observed CRC from the input string, then the last 12 bits of the last line of the intermediate output will be the computed CRC.  However, if you include the observed CRC in the dividend (and therefore do not add padding), then the *calculated* CRC will be the XOR of the *observed* CRC with the last 12 bits of the last line of the intermediate results output. You will therefore note that if those last 12 bits are all zeroes, then this value will be equal to the observed CRC, from which you will be able to conclude that the CRC check was successful (a "Yes" final result).

Regardless of the method chosen for performing the division in verify mode, the final answer must be the result of comparing the computed CRC with the observed CRC.

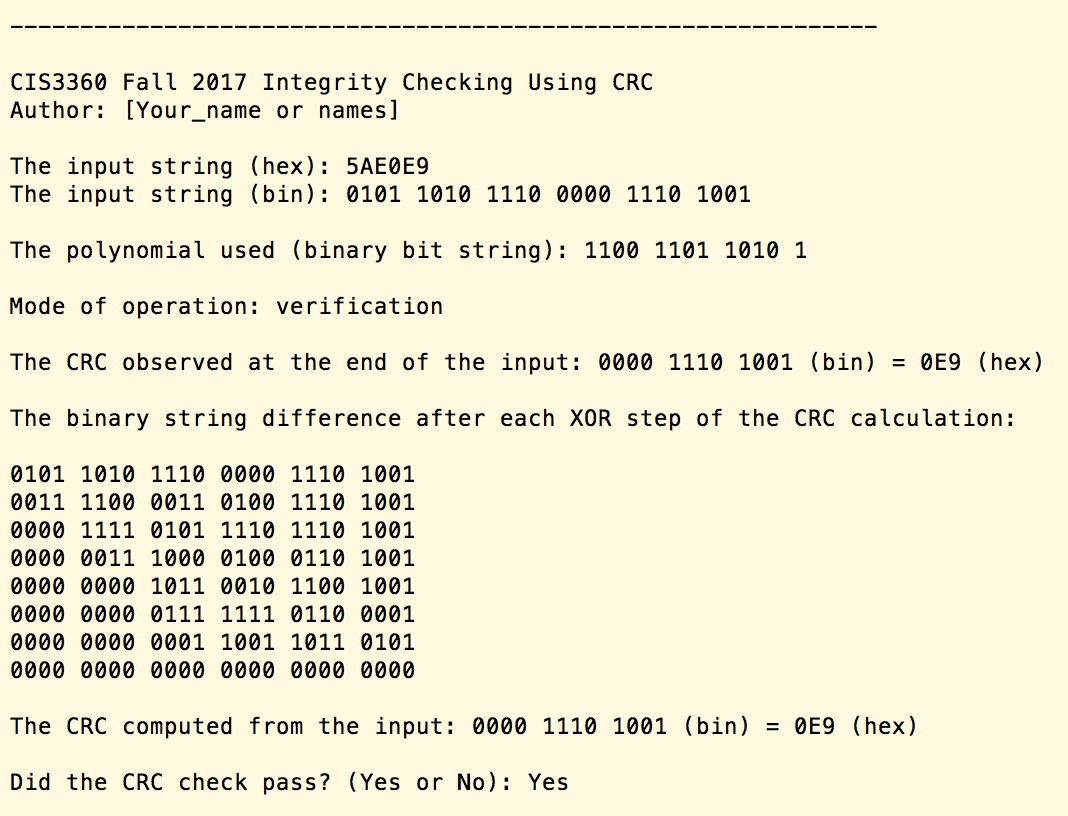
**Sample Outputs**

This section includes three sample outputs, one for each possible situation. *Please note that the verification mode intermediate results show the values for the method where the observed CRC is included in the dividend and no zeroes are added for padding.* If the other method for division in verification mode is used, where the observed CRC is removed and zeroes are added for padding, the intermediate results strings for verification mode would appear exactly the same as in the first example for calculate mode, since the dividends would be identical in both cases.

***(1) Calculation mode output for the command "CrcIntegrity c 5AE":***



***(2) Verification mode output for the command "CrcIntegrity v 5AE0E9".  In this case, the verification is successful:***



***(3) Verification mode output for the command "CrcIntegrity v 5AE91C". In this case, the verification is not successful:***

