Program 2

Due Jul 24, 2015 by 11:59pm **Points** 100 **Submitting** a file upload
File Types c, cpp, and java

CRC Checker

For this assignment you will write a CRC checker program that has two modes of operation and implements the CRC calculation method that we have discussed in lecture. In the calculation mode, the program will calculate the CRC-12 value for a given input string. In the verification mode, the program will detect a CRC-12 value at the end of the input string and then determine whether this is the correct CRC-12 value for the input string when the given CRC value is removed. 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 $x^{12} + x^{10} + x^7 + x^5 + x^3$.

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 Standard Template Library. 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.

What You Should Submit

You should submit a single source code file for one of the permitted languages. Multiple submissions are permitted, but only the last submission before the deadline will be graded. 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 package statements so our test scripts can run without changes.

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

Your program source file should have a header identifying you as the program author. The header should use the following form:

//---

// University of Central Florida

// CIS3360 - Summer 2015

// Program Author: [your_name]

//---

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.

Command Line Parameters

The program must read in two command line arguments or parameters. If you are unsure what is meant by this, or how to use them, 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. 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. For this assignment, you will only need to read in two strings.

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 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 parameters, 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 calculating a CRC value, or “v” for verifying a CRC value (without the 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 be a valid sequence of uppercase hexadecimal characters. There may be as few as 3 or as many as 40 characters in this string. In verify mode, the last 3 hex characters will be 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 CrcChecker [your_file_name].c

C++ program: prompt> g++ -lm -o CrcChecker [your_file_name].cpp

Java program: prompt> javac CrcChecker.java
```

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

```
C/C++ program in Windows: prompt> CrcChecker [mode] [string_to_process]

C/C++ program in Linux: prompt> ./CrcChecker [mode] [string_to_process]

Java program on all systems: prompt> java CrcChecker [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: "CrcChecker c A2B" 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 parameters. 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 free access to the 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 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 may find it convenient to 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) to which you will need to add 12 additional zero values as padding to start off the computation.

The method discussed in lecture is illustrated using the following example in which we compute a CRC-3 code for the message 5AE, using the polynomial $x^3 + x^2 + 1$. 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 4-bit string 1101. Next, we convert the input string 5AE to its binary equivalent, 010110101110. Next, we pad this value with 3 zeroes at the right since we are performing CRC-3 in this example, producing the value 010110101110000, which we call the dividend. This is the value that we divide by the polynomial. The remainder that we get from this division will be the desired CRC-3 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, 100, shown in red at the bottom. Since this is a 3-bit binary number, if we wish to express it as a hex character we will need to pad it with a leading zero to get the 4-bit binary number 0100 (which is the same number), which is of course the hex character 4.

	1	1	0	1	|	0	1	0	1	1	0	1	0	1	1	1	0	0	0	0
⊕							1	1	0	1										
								1	1	0	0									
⊕								1	1	0	1									
											1	1	0	1						
⊕											1	1	0	1						
															1	1	0	0		
⊕															1	1	0	1		
																		1	0	0

Now, 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 15, which is computed from the length of the binary input (3 hex characters = 12 bit), plus the number of pad characters (here, 3). So, we need to report the results of the XOR operations as 15-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, with the desired 15-bit output lines shown in bold.

	1	1	0	1	|	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
⊕							1	1	0	1										
						0	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
⊕								1	1	0	1									
						0	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
⊕											1	1	0	1						
						0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**
⊕															1	1	0	1		
						0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**

Programming Tips

In developing your programs, you may 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 in only 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 the string "CRC Checker by ", followed by your name and then a newline.

2. The program must write the string "Mode of operation: ", followed by either the word "calculate" or "verify", depending on the value of the mode command line input argument

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

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

5. The program must write the string "The polynomial that was 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.

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

   a. If in calculate mode, this line should read: "We will append 12 zeroes at the end of the binary input."

   b. If in verify mode, this line should read: "The 12-bit 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 this statement, this section will show on separate lines the results of each XOR step as described above, with spaces every 4 characters to improve readability.

Please note that for calculate mode, you must pad the binary version of the input string with 12 zeroes before you start the division process, but for verify 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 calculate mode, the final results section will consist of the string "The computed CRC for the input is: ", 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 verify mode, the final results section will consist of two lines. The first line will reporting the computed CRC in binary and hex form, in the same format as for calculate 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: Determining the calculated CRC in verify mode is different, depending on whether the observed CRC was stripped from or included in the dividend, as described in the previous section. If the observed CRC was stripped from the input string, then the last 12 bits of the last line of the intermediate output will be the computed CRC. However, if the observed CRC was included in the dividend 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 had been 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 that case.

(1) Calculate mode output for the command "CrcChecker c A2B":

```
CRC Checker by John Doe

Mode of operation: calculate
The input string (hex): A2B
The input string (bin): 1010 0010 1011

The polynomial that was used (binary bit string): 1010 0101 0100 0
We will append 12 zeroes at the end of the binary input.

The binary string difference after each XOR step of the CRC calculation:

1010 0010 1011 0000 0000 0000
0000 0111 1111 0000 0000 0000
0000 0010 1101 1010 0000 0000
0000 0000 0100 1111 0000 0000
0000 0000 0001 1101 1010 0000
0000 0000 0000 1001 0000 1000

The computed CRC for the input is: 1001 0000 1000 (bin) = 908 (hex)
```

(2) Verify mode output for the command "CrcChecker v A2B908". In this case, the verification is successful:

```
CRC Checker by John Doe

Mode of operation: verify
The input string (hex): A2B908
The input string (bin): 1010 0010 1011 1001 0000 1000

The polynomial that was used (binary bit string): 1010 0101 0100 0
The 12-bit CRC observed at the end of the input: 1001 0000 1000 (bin) = 908 (hex)

The binary string difference after each XOR step of the CRC calculation:

1010 0010 1011 1001 0000 1000
0000 0111 1111 1001 0000 1000
0000 0010 1101 0011 0000 1000
0000 0000 0100 0110 0000 1000
0000 0000 0001 0100 1010 1000
0000 0000 0000 0000 0000 0000

The computed CRC for the input is: 1001 0000 1000 (bin) = 908 (hex)

Did the CRC check pass? (Yes or No): Yes
```

(3) Verify mode output for the command "CrcChecker v A2B627". In this case, the verification is not successful:

```
CRC Checker by John Doe

Mode of operation: verify
The input string (hex): A2B627
The input string (bin): 1010 0010 1011 0110 0010 0111

The polynomial that was used (binary bit string): 1010 0101 0100 0
The 12-bit CRC observed at the end of the input: 0110 0010 0111 (bin) = 627 (hex)

The binary string difference after each XOR step of the CRC calculation:

1010 0010 1011 0110 0010 0111
0000 0111 1111 0110 0010 0111
0000 0010 1101 1100 0010 0111
0000 0000 0100 1001 0010 0111
0000 0000 0001 1011 1000 0111
0000 0000 0000 1111 0010 1111

The computed CRC for the input is: 1001 0000 1000 (bin) = 908 (hex)

Did the CRC check pass? (Yes or No): No
```

Some Rubric (1)

Criteria	Ratings		Pts
The program source code contains the required header and compiles and runs without crashing for all test cases	20.0 pts Full Marks	0.0 pts No Marks	20.0 pts
For all test cases, program output includes the correct title with student name, mode, and input string in both hex and binary format	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
For all test cases, program output includes the correct binary form for the polynomial and also the required padding statement for calculate mode or the correct statement with observed CRC in verify mode	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
For all test cases, program output contains correct intermediate results outputs from XOR calculations, with spaces every 4 characters for readability; in verify mode, the starting value can include the observed CRC (with no additional padding)	20.0 pts Full Marks	0.0 pts No Marks	20.0 pts
For all calculate mode test cases, the program reports the correct calculated CRC value	20.0 pts Full Marks	0.0 pts No Marks	20.0 pts
For all verify mode test cases, the program reports the correct calculated CRC value and also correctly reports whether the verification is successful	20.0 pts Full Marks	0.0 pts No Marks	20.0 pts
			Total Points: 100.0