EE450 Programming
Assignment 1 Spring 2021

Due Date: Sunday September 26th, 2021 11:59 PM (Midnight)

Hard Deadline (Strictly Enforced: the submission website will automatically close on the deadline)

The objective of this assignment is to help you get familiar with error detection algorithm (i.e. cyclic redundancy check, CRC). **It is an individual assignment and no collaborations are allowed. Any cheating will result in an automatic F in the course (not just in the assignment).** If you have any doubts/questions, email TA your questions or come by TA's office hours. **You can ask TAs any question about the content of the project, but TAs has the right to reject your request for debugging.**

Problem Statement:

When you send a sequence of bits (message) from one point to another on a datalink, you want to know that the message arrived correctly. A common form of insurance is to use a parity bit scheme. Most communications protocols use a multibit generalization of the single parity bit called a "**cyclic redundancy check**" or **CRC**. Such an X-bit CRC has the mathematical property of detecting all errors that occur in X or fewer consecutive bits, for any length of message. A single CRC can be associated with a large block of data, with a consequent savings in communications bandwidth.

Another idea is to use **checksum**. One simple version is **one-byte version of internet checksum**. As an example, the ASCII number of 'EE450' is [69, 69, 52, 53, 48]. The one-byte- checksum method will sum these numbers, divide it by 256. As a result, the remainder(R) is 35 and quotient(Q) is 1. Add Q to R and then take 1's complement of that number which is same as subtracting it from 255. the **one-byte version of internet checksum** is 219. If the sum(Q+R) exceeds 255, then divide the sum (Q+R) by 256. With new quotient Q' and remainder R', add Q' to R', take 1's complement of the number to get the **one-byte version of internet checksum**. Once some errors happen, 'EE450' changes to 'E?450' which corresponds to [69, 33, 52, 53, 48]. Suppose the checksum does not change, the receiver proceeds the similar process. The receiver finds the resulting checksum (0) mismatch with the checksum (219) received. It decides not to accept that data. However, in some cases, the performance of checksum may be worse than that of CRC. **Why? You will be asked to figure that out later.**

Part 1: CRC_Tx

In this project, you are required to implement **CRC-12** both at the transmitter (Tx.) side and the receiver (Rx.) side. At transmitter side, **CRC_Tx** will read data and the corresponding generator from a **dataTx.txt** file provided. The file contains several rows of **data** separated by a new line. After CRC implementation, CRC_Tx will printout generated codeword and corresponding CRC bits to add. You will use the standard CRC-12 generator: $x^{12} + x^{11} + x^3 + x^2 + x + 1$ = 1100000001111.

Example input (from dataTx.txt):

10011010110101

Example output:
codeword: 100110101101011111100001100
crc: 111100001100

Part 2: CRC_Rx

At the receiver side, **CRC_Rx** will read received codeword from **dataRx.txt** provided. Then CRC_Rx will printout the decision whether the received data should be accepted (Pass) or not (Not pass). Used the same CRC-12 generator as part 1.

Example input (from dataRx.txt):
100110101101011111100001100

Example output:
pass

Part 3: CRC_vs_Checksum

The last step will involve using both the transmitter and receiver steps, **CRC_vs_Checksum** will be programmed to compare the performance of CRC and checksum for some example data. CRC_vs_Checksum will read data from dataVs.txt which contains three parts: data, generator, and random bits error. It will then proceed to do the following:

1. Calculate the CRC and checksum for each data (each row in dataVs.txt will contain a new set of data).
2. Calculate the CRC and checksum bits
3. Append the CRC and checksum to the data
4. To introduce the random bits error, do XOR operation to the encoded data (random bits error has the same length as generated codeword: data+CRC and data+checksum, if it is 0, there is no error at the position; if it is 1, the bit of the codeword will be flipped).
5. Check the resulting data with CRC and checksum
6. **Print** the CRC and Checksum bits for that row.
7. **Print** whether the data would be accepted or not (pass vs not pass) for both CRC and Checksum.

How errors will be introduced:

Codeword:	100110101101011111100001100
Error bits from input file:	000010000010000000000001000
Codeword with errors (red highlights show errors):	100100101111010111100000100

**See the example output below in screenshots.

Source Code Files

Your implementation should include the source code files described below, for each component of the system.

1. CRC_Tx: You must name your code file: crc_tx.c or crc_tx.cc or crc_tx.cpp (all small letters). Also you must call the corresponding header file (if you have one; it is not mandatory) crc_tx.h (all small letters).
2. CRC_Rx: You must name your code file: crc_rx.c or crc_rx.cc or crc_rx.cpp (all small letters). Also you must call the corresponding header file (if you have one; it is not mandatory) crc_rx.h (all small letters).
3. CRC_vs_Checksum: You must name your code file: crc_vs_checksum.c or crc_vs_checksum.cc or crc_vs_checksum.cpp (all small letters). Also you must call the corresponding header file (if you have one; it is not mandatory) crc_vs_checksum.h (all small letters).

Example Output to Illustrate Output Formatting:

```
ee450@ee450-VirtualBox:~/Documents/pa1/PA1/final$ make all
g++ -std=c++0x -o crc_tx crc_tx.cpp
g++ -std=c++0x -o crc_rx crc_rx.cpp
g++ -std=c++0x -o crc_checksum crc_checksum.cpp
./crc_tx
codeword:
1111010100000011000101001010001101011111000111001111111011111011101010000011100
0010101001100110001011110010 0
crc:
010111100100
codeword:
010101101011000110011010 0
crc:
001100110100
codeword:
1111110101000001101001011101110000000000001000011111010011000110101101010011110
10011111010111000
crc:
111010111000
codeword:
1101011100100000001001010100110001001011000110010001000000001001011001000011000
101
crc:
000110000101
codeword:
1100011110001010011010110000101100010000110110000010110000
crc:
000010110000
```

```
./crc_rx
pass
not pass
pass
not pass
pass
not pass
pass
not pass
pass
not pass
```

```
./crc_checksum
crc : 111101011010  result: not pass
checksum: 00001110  result: not pass

crc : 101110100011  result: not pass
checksum: 01101000  result: pass

crc : 010000010001  result: pass
checksum: 01100110  result: not pass

crc : 100111111110  result: not pass
checksum: 00001011  result: not pass

crc : 001101110010  result: not pass
checksum: 01101100  result: not pass

crc : 111001110100  result: not pass
checksum: 10101110  result: pass

crc : 101010010110  result: not pass
checksum: 01110111  result: not pass

crc : 011110110101  result: not pass
checksum: 11110001  result: not pass

crc : 000010111100  result: not pass
checksum: 01010111  result: not pass

crc : 001101001001  result: not pass
checksum: 00110100  result: not pass

crc : 011101110111  result: pass
checksum: 11001101  result: pass
```

Assumptions:

If you need to have more code files than the ones that are mentioned here, please use meaningful names and all small letters and mention them all in your README file.

Requirements:

1. Your programs should terminate itself after all done.
2. All the naming conventions and the on-screen messages must conform to the previously mentioned rules.
3. All the on-screen messages must conform exactly to the project description. You should not add anymore on-screen messages. If you need to do so for the debugging purposes, you must comment out all the extra messages before you submit your project.

Programming platform and environment:

1. All your submitted code **MUST** work well on the provided virtual machine Ubuntu.
2. All submissions will only be graded on the provided Ubuntu. TAs won't make any updates or changes to the virtual machine. It's your responsibility to make sure your code working well on the provided Ubuntu. "It works well on my machine" is not an excuse and we don't care.
3. Your submission MUST have a Makefile. Please follow the requirements in the following "Submission Rules" section.

Programming languages and compilers:

You must use only C/C++ on UNIX.

You can use a unix text editor like emacs or gedit to type your code and then use compilers such as g++ (for C++) and gcc (for C) that are already installed on Ubuntu to compile your code. You must use the following commands and switches to compile yourfile.c or yourfile.cpp. It will make an executable by the name of "yourfileoutput".

gcc -o yourfileoutput yourfile.c g++ -o yourfileoutput yourfile.cpp

Do NOT forget the naming conventions mentioned before!

Submission Rules:

Along with your code files, include a **README file and a Makefile**. In the README file write

1) Your Full Name as given in the class list
2) Your Student ID
3) What you have done in the assignment.
4) What your code files are and what each one of them does. (Please do not repeat the project description, just name your code files and briefly mention what they do).

5) Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
6) Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code.)

Submissions without README and Makefile will be subject to a penalty.

Makefile tutorial:

https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html

About the Makefile:

makefile should support following functions:

make all	Compile **all** your files, create executable files, and run all three executables
./crc_tx	**Run** CRC_Tx
./crc_rx	**Run** CRC_Rx
./crc_vs_checksum	**Run** CRC_vs_Checksum

** We will spend time discussing Makefiles in the 9/20 discussion session if this is your first time!

TA will compile and run all codes using **make all.**

Other details for turning in the assignment:

1. Compress all your files including the README file into a single "tar ball" and call it: **ee450_PA1_yourUSCusername.tar.gz**. Please make sure that your name matches the one in the class list. Here are the instructions:

   a. On your VM, go to the directory which has all your project files. Remove all executable and other unnecessary files. **Only include the required source code files, Makefile and the README file**. Now run the following commands:

   b. Use these commands in the terminal to compress your files into a tar ball"

>> tar cvf **ee450_PA1_yourUSCusername.tar** *

>> gzip **ee450_PA1_yourUSCusername.tar**

Now, you will find a file named "ee450_PA1_yourUSCusername.tar.gz" in the same directory. Please notice there is a star(*) at the end of first command.

c. Do NOT include anything not required in your tar.gz file. Do NOT use subfolders. **Any compressed format other than .tar.gz will NOT be graded!**

2. Submit your ee450_PA1_yourUSCusername.tar.gz file to blackboard before the submission timeline. You can submit as many times as you want, only the latest submission (before the deadline) will be considered while grading.

3. Please consider all kinds of possible technical issues and do expect a huge traffic on the blackboard website very close to the deadline which may render your submission or even access to blackboard unsuccessful. Please DO NOT wait till the last 5 minutes to upload and submit because some technical issues might happen, and you will miss the deadline.

4. After receiving the confirmation email, please confirm your submission by downloading and compiling it on your machine. If the outcome is not what you expected, try to resubmit, and confirm again. We will only grade what you submitted even though it's corrupted.

Grading Criteria:

Notice: We will only grade what is already done by the program instead of what will be done.

Your project is graded for 100 points and your grade will depend on the following:

1. Correct functionality.

2. Inline comments in your code. This is important as this will help in understanding what you have done.

3. Whether your programs work as you say they would in the README file.

4. If your submitted codes, cannot be compiled, you will receive **10 out of 100** for the project, assuming submission includes README file.

5. If your submitted codes compile using make but when executed, produce runtime errors without performing any tasks of the project, you will receive **15 out of 100** for the project.

6. If you forget to include the README file or Makefile in the project tar-ball that you submitted, you will lose **15 points** for each missing file (plus you need to send the file to the TA in order for your project to be graded.)

7. Do not submit datafile (three .txt files) used for test, otherwise, you will lose **5 points**.

8. The minimum grade for an on-time submitted project is **10 out of 100**, assuming there are no compilation errors and the submission includes a working Makefile and a README.

9. There are no points for the effort or the time you spend working on the project or reading the tutorial. If you spend about 2 weeks on this project and it doesn't even compile, you will receive only **10 out of 100**.

10. Your code will not be altered in any way for grading purposes and however it will be tested with different inputs. Your TA runs your project as is, according to the project description and your README file and then check whether it works correctly or not. If your README is not consistent with the description, we will follow the description.

Cautionary Words:

In view of what is a recurring complaint near the end of a project, we want to make it clear that the target platform on which the project is supposed to run is *the provided **Ubuntu (16.04)***. It is strongly recommended that students develop their code on this virtual machine. In case students wish to develop their programs on their personal machines, possibly running other operating systems, they are expected to deal with technical and incompatibility issues (on their own) to ensure that the final project compiles and runs on the requested virtual machine. If you do development on your own machine, please leave at least three days to make it work on Ubuntu. It might take much longer than you expect because of some incompatibility issues.

Academic Integrity:

All students are expected to write all their code on their own.

Copying code from friends is called **plagiarism** not **collaboration** and will result in an F for the entire course. Any libraries or pieces of code that you use, and you did not write must be listed in your README file. All programs will be compared with automated tools to detect similarities; examples of code copying will get an F for the course. **IF YOU HAVE ANY QUESTIONS ABOUT**

WHAT IS OR ISN'T ALLOWED ABOUT PLAGIARISM, TALK TO THE TA. "I didn't know" is not an excuse.