EE450 Programming Assignment 1

Fall 2019

Due Date: Sunday September 29th, 2019 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). This assignment is worth 3% of your overall grade in this course. **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.

One simplier 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 remainer(R) is 35 nd 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.**

In this project, you are required to implement CRC both at the transmitter (Tx.) side and the receiver (Rx.) side. Also you are required to compare the performance of CRC and checksum under certain circumstances.

At transmitter side, **CRC_Tx** will read data and the corresponding generator from a **data2Tx.txt** file provided. The file contains two parts: **data** and **generator**. After CRC implementation, CRC_Tx will printout generated codeword and corresponding CRC bits to add.

At receiver side, **CRC_Rx** will read received codeword and the corresponding generator from

data2Rx.txt proviced. Then CRC_Rx will printout the decision whether the received data should be accepted (Pass) or not (Not pass).

To further investigate the performance of CRC, **CRCvsChecksum** will be programmed to compare the error detection rate of CRC and checksum. CRCvsChecksum will read data from **data2Vs.txt** file. The file contains three parts: **data**, **generator**,**random bits error**. The CRCvsChecksum should proceed in the following:1) **calculate** CRC and checksum for each data; 2) **append** the CRC and checksum to the data; 3) to introduce the ramdom bits error, do XOR operation to the encoded data(**random bits error** has the same length as generated codeword: data+CRC/checksum, if it is **0**, there is no error at the position; if it is 1, the bit of the codeword will be flipped); 4) check the resulting data with CRC and checksum; 5) compare the performance.

To compare the performance, the F1-measure will be calculated. F1-measure is calculated defined as $F1 = 2\,\frac{Precision \times Recall}{Precision+Recall}$, where $Precision$ is defined as $\frac{True\ positive}{True\ positive+False\ positive}$ and $Recall$ is defined as $\frac{True\ positive}{True\ positive+False\ negative}$. Generally speaking, higher the F1-score, better the performance.

		Receiver Decision	
		Not pass	Pass
Actual	Error	True negative	False positive
	No Error	False negative	True positive

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. CRCvsChecksum: 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_rx.h(all small letters).

Example Output to Illustrate Output Formatting:

CRC_Tx:

```
codeword:
10111011
crc:
011
codeword:
101010011100
crc:
1100
codeword:
10101001010011010010110101111101110000010110110011011001110000110000111 1
crc:
00001111
```

CRC_Rx:

```
pass
pass
Not pass
Not pass
```

CRCvsChecksum:

```
F1 socre of 1-byte checksum:0.688525
F1 socre of CRC:0.976744
```

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 of 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 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 mandatory naming conventions mentioned before!

Submission Rules:

1. Along with your code files, include a **README file and a Makefile**. In the README file write
   a. Your **Full Name** as given in the class list
   b. Your Student ID
   c. What you have done in the assignment.
   d. 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).
   g. Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
   h. 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 SERIOUS PENALTY

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

About the Makefile: makefile should support following functions:

`make all`	Compiles **all** your files and creates executables
`make crc_tx`	**Run**s CRC_Tx
`make crc_rx`	**Run**s CRC_Rx
`make crc_vs_checksum`	**Run**s CRC vs Checksum

TA will first compile all codes using **`make all`** and run each program with the corresponding makefile command.

2. 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.
   >> `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!

3. Please take into account all kinds of possible technical issues and do expect a huge traffic on the DEN website very close to the deadline which may render your submission or even access to DEN unsuccessful.

4. 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. And a kind suggestion, if you still get some bugs one hour before the deadline, please make a submission first to make sure you will get some points for your hard work!

5. 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 ponts 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.

5. If your submitted codes compile using make but when executed, produce runtime errors without performing any tasks of the project, you will receive **10 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 get **20 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 **5 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.