SSRL FSW Interview Fall 2022

Prepared by: Cameron Bonesteel

Due: 12 October 2022

Introduction

Welcome to the Flight Software (FSW) take home interview for the Fall 2022 recruitment cycle! This software project is the last thing between you and an acceptance to the lab, so be sure to put in good work! To get you started, I want you to understand that I am not looking for perfection, completion, or fancy optimized code. What I am looking for is an understanding of the problem, a solid problem solving approach, good research practices, and an attempt to learn the new concepts in this project. If you do all of these things and express it thoroughly in your README (as discussed later in this document) you will have a good chance at acceptance. If you do no research, copy code directly from stack overflow without citing it, or fail to write any code, you will certainly fail. With that being said, if you have any questions, please email to me at **cameron.bonesteel@uga.edu** and if I deem the question important to increase clarity of the problem without giving away the answer, all applications will receive an update email with the clarification. Best of luck!

Task 1 (of 2): Communication Breakdown

Introduction

Packetization of data is a very important concept in the world of networks but it has great applications to space too! Packetization of data allows data to be transferred from computer to computer via networks or radio frequencies (RF), which is our application, while simultaneously checking the data for corruption. A large file or other data can be split into smaller packets, which is then defined with a header and a cyclic redundancy check (CRC). The header defines important information about the packet, in this case, the Packet ID, the total number of packets, and the length of the data present in the packet. This information is generated at the time of the packet generation based on the file being packetized and the data present. The CRC for this problem is defined as the following:

```
The one's complement of the sum of the bytes of the packet,
including all header information and packet data.
```

This sum is calculated at the time of packet generation, sent with the packet after the data, and is then calculated again by the receiving system and compared to the sum in the packet. If these values are different, either the header, data, or the sum corrupted during transmission.

Problem Definition

You will be given two binary files, one with complete packets, and one with an unknown number of corrupted packets. You are to write a packet decoding program that decodes the binary files based on the packet definitions below. This program should extract the data from the header and tail and place it into a new file so long as the CRC is correct for the packet. For the complete binary, this should produce an encoded script for a movie. We will decode this in the second task. For the corrupted binary, this should produce a list of corrupted packets using the packet id. Your program should take one command-line argument, the filename to be decoded.

The packet is defined as follows:

```
Packet ID:                          2 bytes
Number of Packets:                  2 bytes
Length of the Data in the Packet:   2 bytes
Data:                               maximum 1016 bytes
CRC:                                2 bytes
```

Remember: A packet is only valid if the CRC in the packet is equivalent to the calculated CRC, where the CRC is the one's compliment of the sum of all header and packet data.
Hint: A packet's data may be up to 1016 bytes but may be shorter.

Deliverable

You may approach this problem in any programming language, however, our flight software tasks use C/C++ so using these languages is the recommended approach.

Your deliverables are:

- A program that will parse packets and compile the data into one new file as long as the CRC is valid. The program should take one command line argument, the file to be decoded.
- The extracted file from the complete binary file.
- The list of corrupted packets for the corrupted binary file.
- Sufficient documentation and notes so that another developer can easily determine how your functions work.
- A README that has notes on how you approached the problem, resources you used and how you solved it, and any difficulties that you had while completing the problem.
- A Makefile (if using a compiled language) that can compile the program.

Task 2 (of 2): Caesar Cipher

Introduction

The Caesar Cipher is a decades old cipher that is easy to use and applies well to computer science. Each character is shifted by a certain number of places in the alphabet or character set. Since ASCII characters are just represented by numbers, shifting these characters is very easy. We used a Caesar Cipher to encrypt our movie script, so now that you have extracted the script from the packets, we need to decipher it.

Problem Definition

We have encrypted our movie script by shifting the entire ASCII character set by 8 characters. Using this key, decipher the movie script and put it back into it's human readable form.

Deliverable

You may approach this problem in any programming language, however, our flight software tasks use C/C++ so using these languages is the recommended approach.

Your deliverables are:

- A program that decipher the movie script. It should take one command line argument, the file to be deciphered.
- The deciphered movie script.
- Sufficient documentation and notes so that another developer can easily determine how your functions work.
- A README that has notes on how you approached the problem, resources you used and how you solved it, and any difficulties that you had while completing the problem.
- A Makefile (if using a compiled language) that can compile the program.

Submission Instructions

Submit a tarball that includes the following via email to **cameron.bonesteel@uga.edu**:

- A folder named ***Task1*** that includes:
  - Your Program
  - A Makefile (If using a compiled language)
  - The decoded file
  - A list of corrupted packets
  - A README (.md or .pdf)
- A folder named ***Task2*** that includes:
  - Your Program
  - A Makefile (If using a compiled language)
  - The deciphered script
  - A README (.md or .pdf)

Use the following command to compress the parent folder into a tarball:

```
tar -zcvf LastName_FirstName_FSWInterview.tar.gz [Your_Folder]
```