EECS 293

Software Craftsmanship

2020 Spring Semester

Programming Assignment 9

Due at the beginning of your discussion session on

March 24-27, 2020

Reading

In addition to the following topics, the quiz syllabus includes any material covered in the lectures: Sections 19.6 in Code Complete and the Quick Reference on Routine Names on canvas.

Chapter 23 in Code Complete will help you with the programming assignment but it is not part of the quiz.

Grading Guidelines

For programming assignment 9 only, *late assignments* will be penalized by 20% instead of the usual 10%.

Points will be deducted if code and branch coverage is incomplete. An automatic C (or less) is triggered by:

- Any routine with complexity greater than 4,
- Any substantially repeated piece of code, or by
- Improperly named routines.

Programming

In this assignment, you will debug an application for connecting to a wireless channel (posted on canvas). Here are the original specifications, which were then incorrectly implemented in the provided code.

The software scans channels in a secure peer-to-peer network. Its objective is for two wireless interfaces to agree on a channel to use for communication. In short, each radio scans through the channels, pausing briefly on every channel on which communication is detected. If it hears a call from another radio to its own address, it will stop on that channel and establish a connection. Although the protocol is encrypted, in this assignment, you can assume that the data stream has already been decrypted and presented to your protocol in plaintext. Your component listens to a channel and decides whether to connect or not.

Every wireless interface has an address, which is a byte array. The address length is unspecified and can vary across interfaces. To establish a call, the caller will transmit a sequence that starts with the address of the intended receiver followed by the address of the caller. The simplest type of call is one between two interfaces whose address is a single digit, and looks like:

```
TO 1
TO 1
THISIS 4
```

The sequence represents a call from interface 4 to interface 1. The recipient address is repeated exactly twice. If either endpoint has a longer address, a more complex sequence is used:

```
TO 1
REP 2
REP 3
TO 1
REP 2
REP 3
THISIS 4
REP 5
```

The sequence represents a call from interface 45 to interface 123.

There is an additional problem, though. Assume that 45 wants to establish a connection to 123. Interface 123 is repeatedly scanning through many channels, spending a few milliseconds on each channel to see if there is an incoming call on that channel. If 45 wants to use a channel for the call, it must transmit long enough on that channel so that radio 123 has time to go through all channels and still be in time to hear the call from 45. To solve this problem, radio 45 has the option to repeat the first sequence element (TO) thus giving radio 123 the opportunity to hear the first digit of its address. Other interfaces like 1 and 167 will also stop to listen but they will continue scanning once the full address follows. Then, the calling sequence becomes:

```
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
TO 1
REP 2
REP 3
TO 1
REP 2
REP 3
THISIS 4
REP 5
```

The initial `TO` can be repeated between one and ten times, and the exact number of repetitions is at the discretion of the caller. The remaining `TO` and `REP` are not repeated.

The component input is a receiver address, a local address, and an `Input Stream` and returns both a boolean variable that denotes whether to connect on this channel and the caller address. The input stream is supposed to contain repeated transmissions of the calling sequence as detected on a channel. However, since the receiver may tune in halfway through the transmission, the stream may start part way through the calling sequence. Furthermore, due to electromagnetic interference, some lines may be garbled. The component should return at the end of the first useful calling sequence or at the end of the input, whichever comes first. The code should process the stream incrementally without waiting for the end of the input. The component should be robust and connect even in the presence of various communication errors.

◎

If the provided code fails to implement this specification, you should consider it a defect to fix.

Debug the code provided in canvas to match the specification above. You are not allowed to execute brute-force debugging except in a small number of methods, if at all. Submit a diff file to document all of the fixes that you made. Create test cases that capture the defects that you find. Additionally, if there are test cases missing from the original, even for things you did not change, you should add them. Keep in mind that buggy code can work properly (e.g. unintended/unneeded cyclomatic complexity) and should be fixed.

Group Evaluation

Programming Assignment 9 is a single assignment project. In the next assignment, we will turn to a different project. In addition to the regular submission, you are required to post a private comment on canvas on the group dynamics, and in particular on the percentage effort that you feel was put together by the individual team members.

General Considerations

Your implementation may contain as many auxiliary private methods as you see fit, and additional helper classes may be defined. Your code should have an exhaustive unit test suite. Your code should have a reasonable number of comments, but documentation is going to be the topic of a future assignment. As a general guideline at this stage of the course, comments should be similar to those accepted in EECS 132.

Discussion Guidelines

The class discussion will focus on refactoring (changes to the source code) to fix bugs, and on the debug test cases.

Submission

Create a repository called channels.git where you will post your submission. Make small regular commits and push your revised code and test cases on the git repository.