PS7 Kronos – Intro to Regular Expression

In this assignment, we were to utilize regular expressions to parse Kronos InTouch time clock records. We were to check particular lines in the log with regular expressions and report whether a boot was successful or not in a separate file.

Key concepts

This assignment's code was included in a single source file, main.cpp. For both the regex functionality and interacting with the date and time, I utilized the Boost libraries. The only regex I used was to capture the date and time that each boot begun or boot finished line reported. The boot messages themselves were constant and could be scanned without using regex.

What I Accomplished

Depending on the message scanned, the program scans a log file line by line and decides if a boot has started or completed. The date and time are stored and reported if it contains a boot start message. When a boot finished notification is scanned, the date and time, as well as the total boot duration from start to finish, are stored again. This data is stored in a report file (.rpt) that contains each successful and unsuccessful boot attempt.

What I Learned

Despite the fact that this was a short task, I learned the value of regular expressions and how they may be used when scanning for or extracting string/character data. I also discovered how to retrieve the total boot time between the boot start and boot end events in the log using the Boost date/time library.

```
CC = g++
CFLAGS = -std=c++11 -c -g -O1 -Wall -Werror -pedantic
BOOST = -lboost_regex -lboost_date_time

all: ps7

ps7: main.o
	$(CC) -o ps7 main.o $(BOOST)
main.o: main.cpp
	$(CC) -c main.cpp $(CFLAGS)

clean:
	rm -f *.o ps7 *~
```

```cpp
// Copyright 2022 Matthew Lorette Anaya
#include <iostream>
#include <fstream>
#include <string>
#include <exception>
#include <boost/regex.hpp>
#include <boost/date_time.hpp>

using boost::regex;
using boost::regex_search;
using boost::smatch;
using boost::posix_time::ptime;
using boost::posix_time::time_duration;
using boost::posix_time::time_from_string;

int main(int argc, char* argv[]) {

    int lineNum = 1,
        bootStCnt = 0,
        bootDoneCnt = 0;

    bool bootStarted = false;

    const std::string bootStMsg = "(log.c.166) server started";
    const std::string bootDoneMsg = "oejs.AbstractConnector:Started "
    "SelectChannelConnector@0.0.0.0:9080";
    regex e("^\\d{4}[-](0[1-9]|1[012])[-](0[1-9]|[12][0-9]|3[01])\\s\\d{2}"
    "[:]\\d{2}[:]\\d{2}");
    smatch m;

    ptime tBST, tBDT;

    std::string s;
    std::string fileName;
    std::ifstream inFile;
    std::ofstream outFile;

    if (argc != 2) {
        std::cerr << "Usage: ./ps7 device1_intouch.log" << std::endl;
        return -1;
    }

    inFile.open(argv[1]);
    if (!inFile.is_open()) {
        std::cerr << "Could not open file: " << argv[1] << std::endl;
        return -1;
    }
    s = fileName = argv[1];
    outFile.open(s.append(".rpt.tmp"));
    fileName = fileName.substr(fileName.find_last_of("\\/") + 1);

    // Temp report file = scanned boot
    while (std::getline(inFile, s)) {
        if (bootStarted) {
            if (s.find(bootDoneMsg) != std::string::npos) {
                // Boot Done
                regex_search(s, m, e);
                tBDT = ptime(time_from_string(m[0]));
                time_duration td = tBDT - tBST;

                outFile << lineNum << "(" << fileName << ") " << m[0]
                << " Boot Completed" << std::endl
                << "\tBoot Time: " << td.total_milliseconds() << "ms"
                << std::endl << std::endl;

                bootStarted = false;
                bootDoneCnt++;
            } else if (s.find(bootStMsg) != std::string::npos) {
                // Failed boot
                regex_search(s, m, e);
                tBST = ptime(time_from_string(m[0]));

                outFile << "**** Incomplete boot ****" << std::endl <<
                std::endl
                << "=== Device boot ===" << std::endl
                << lineNum << "(" << fileName << ") " << m[0]
                << " Boot Start" << std::endl;

                bootStCnt++;
            }
        } else if (s.find(bootStMsg) != std::string::npos) {
            // Sucessfull boot
            regex_search(s, m, e);
            tBST = ptime(time_from_string(m[0]));
            outFile << "=== Device boot ===" << std::endl
            << lineNum << "(" << fileName << ") " << m[0]
            << " Boot Start" << std::endl;
            bootStarted = true;
            bootStCnt++;
        }
        lineNum++;
    }
    inFile.close();
    outFile.close();

    // Report file done
    s = argv[1];
    s.append(".rpt");
    outFile.open(s);

    s.append(".tmp");
    inFile.open(s);
    if (!inFile.is_open()) {
        std::cerr << "Could not open file: " << s << std::endl;
        return -1;
    }

    outFile << "Device Boot Report" << std::endl << std::endl
    << "InTouch log file: " << fileName << std::endl
    << "Lines Scanned: " << lineNum - 1 << std::endl << std::endl
    << "Device boot count: initiated: " << bootStCnt << ", completed: "
    << bootDoneCnt << std::endl << std::endl << std::endl;

    outFile << inFile.rdbuf();
    inFile.close();
    outFile.close();

    // remove temp report file
    if (std::remove(s.c_str()) != 0) {
        std::cerr << "Error deleting temp file: " << s << std::endl;
        return -1;
    }

    return 0;
}
```