```
 1: //  Copyright 2015 Zheondre Calcano
 2: //  PS7a
 3: #include <boost/regex.hpp>
 4: #include <boost/date_time.hpp>
 5: #include <exception>
 6: #include <stdexcept>
 7: #include <sstream>
 8: #include <fstream>
 9: #include <iostream>
10: #include <string>
11: #include <vector>
12:
13: using namespace std; //NOLINT
14: using namespace boost; //NOLINT
15:
16: void efname(string &name) { name += ".rpt";}
17:
18: void parse(string fn) {
19:   int linenum, completeboot;
20:   vector< int > holdval;
21:   holdval.push_back(0);
22:   holdval.push_back(0);
23:   holdval.push_back(0);
24:   string ufn, filename, lif, rs, rsa, temp, boottime;
25:   ufn = fn;
26:   efname(fn);
27:   std::fstream outfile;
28:   cout << fn << endl;
29:   outfile.open(fn.c_str(), fstream::out);
30:   rs = ".*log.c.166.*";
31:   rsa = ".*oejs.AbstractConnector:Started SelectChannelConnector.*";
32:   string t = "(\\d{2}):(\\d{2}):(\\d{2})";
33:   //  (\d{2}):(\d{2}):(\d{2})
34:   string tmm = "(\\d{2}):(\\d{2}):(\\d{2})\\.(\\d{3})";
35:   //  (\d{2}):(\d{2}):(\d{2})\.(\d{3})
36:   string gd = "(\\d{4})-(\\d{2})-(\\d{2})";
37:   boottime = "Boot Time: ";
38:   outfile << "Device Boot Repot \n" + ufn + "\n\n";
39:   std::ifstream infile(ufn.c_str());
40:   smatch sm, sn, so, sp;
41:   regex e = regex(rs);
42:   regex ea = regex(rsa);
43:   regex etime(t);
44:   regex f(tmm);
45:   regex getdate(gd);
46:   regex getdatea(gd);
47:   std::ostringstream ss;
48:   linenum = completeboot = 0;
49:   while (getline(infile, lif)) {
50:     linenum++;
51:     if (regex_match(lif, e)) {
52:       if (completeboot == 1) {
53:         outfile << "**** Incomplete boot ****\n\n";
54:         completeboot = 0;
55:       }
56:       outfile << "=== Device boot ===\n";
57:       regex_search(lif, sm, etime);
58:       regex_search(lif, so, getdate);
59:       holdval[0] = boost::lexical_cast<int>(sm[1]);
60:       holdval[1] = boost::lexical_cast<int>(sm[2]);
61:       holdval[2] = boost::lexical_cast<int>(sm[3]);
```

```
   62:       ss.str("");
   63:       ss << linenum;
   64:       temp = ss.str();
   65:       temp += "(" + ufn + "):";
   66:       temp += so[0] + " " + sm[0] + " Boot Start \n";
   67:       outfile << temp;
   68:       completeboot = 1;
   69:       temp.clear();
   70:     }
   71:     if (regex_match(lif, ea)) {
   72:       ss.str("");
   73:       ss << linenum;
   74:       temp = ss.str();
   75:       temp += "(" + ufn + "):";
   76:       regex_search(lif, sn, f);
   77:       regex_search(lif, sp, getdatea);
   78:       boost::posix_time::time_duration ta(holdval[0], holdval[1], holdval[2]
);
   79:       boost::posix_time::time_duration tb(boost::lexical_cast<int>(sn[1]),
   80:                                           boost::lexical_cast<int>(sn[2]),
   81:                                           boost::lexical_cast<int>(sn[3]));
   82:       //  tb += boost::posix_time::millisec(boost::lexical_cast<int>(sn[4]))
;
   83:       tb = tb - ta;
   84:       temp += sp[0] + " " + sn[0] + " " + "Boot Completed \n";
   85:       outfile << temp;
   86:       ss.str("");
   87:       ss << tb.total_milliseconds();
   88:       outfile <<"\t"+ boottime + ss.str() + " ms\n\n";
   89:       completeboot = 0;
   90:       temp.clear();
   91:     }
   92:   }
   93:   outfile.close();
   94: }
   95: int main(int argc, char *argv[]) {
   96:   string filename;
   97:   filename = argv[1];
   98:   if (filename.size() < 1)
   99:     throw
  100:       std::runtime_error("Null string for file name");
  101:   parse(filename);
  102:   return 0;
  103: }
```

```
 1: #include <iostream>
 2: #include <fstream>
 3: #include <string>
 4: using namespace std;
 5: /*
 6: int main () {
 7:   ofstream myfile;
 8:   myfile.open ("example.txt");
 9:   myfile << "Writing this to a file.\n";
10:   myfile.close();
11:   return 0;
12: }
13: */
14: void efname( string &name ) { name += ".out";}
15: void wtof( string name, string info){
16:   std::ofstream outfile;
17:   outfile.open(name, std::ios_base::app);
18:   outfile << info;
19: }
20: void parse(string fn ){
21:   int linenum, completeboot, he, hf, hg;;
22:   int vector<int> holdval;
23:   holdval.push_back(0);
24:   holdval.push_back(0);
25:   holdval.push_back(0);
26:   string ufn, filename, lif, rs, rsa, temp;
27:   ufn = fn;
28:   efname(fn);
29:
30:   rs = ".*log.c.166.*"
31:   rsa = ".*oejs.AbstractConnector:Started SelectChannelConnector.*";
32:   string t = "(\\d{2}):(\\d{2}):(\\d{2})";
33:   //(\d{2}):(\d{2}):(\d{2})
34:   string tmm= "(\\d{2}):(\\d{2}):(\\d{2})\\.(\\d{3})";
35:   //(\d{2}):(\d{2}):(\d{2})\.(\d{3})
36:   string gd = "(\\d{4})-(\\d{2})-(\\d{2})";
37:
38:   boottime= "Boot Time: ";
39:   wtof(fn,"Device Boot Repot \n " + uf + "\n");
40:   std::ifstream infile(fn.c_str());
41:   smatch sm, sn, so,sp;
42:   regex e = regex(rs);
43:   regex ea = regex(rsa);
44:   regex etime(t);
45:   regex f(tmm);
46:   regex getdate(gd);
47:   regex getdatea(gd);
48:
49:   linenum = completeboot = 0;
50:
51:  while(getline(infile, lif)){
52:         linenum++;
53:     if(regex_match(lif,e)){
54:                 if( completeboot == 1){ //badboot
55:                         wtof(fn, "**** Incomplete boot ****\n")
56:                         completeboot = 0;
57:                 } else{ // startboot
58:                         wtof(fn, "=== Device boot ===\n");
59:                         regex_search(lif, sm, etime);
60:                         regex_search(lif, so, getdate);
61:                         holdval[0] = boost::lexical_cast<int>(sm[1]);
```

```
   62:                           holdval[1] = boost::lexical_cast<int>(sm[2]);
   63:                           holdval[2] = boost::lexical_cast<int>(sm[3]);
   64:                           temp = std::to_string(linenum); // linenum
   65:                           temp += "(" + ufn + "):";
   66:                           temp += so[0] + " " + sm[0] + " Boot Start \n";
   67:                           wtof(fn,temp);
   68:                           completeboot = 1;
   69:                           temp = "";
   70:                   }
   71:     }
   72:         if(regex_match(lif,e)){ //good boot
   73:                   temp = std::to_string(linenum); // linenum
   74:                   temp += "(" + ufn + "):";
   75:                   regex_search(lif, sn, f);
   76:                   regex_search(lif, sp, getdatea);
   77:                   //I should put a case to check to see if the vector values a
re
   78:                   //equvalent to 0 but I ran out of time.
   79:                   boost::posix_time::time_duration ta(holdval[0],holdval[1],ho
ldval[2])
   80:                   boost::posix_time::time_duration tb(boost::lexical_cast<int>
(sn[1]),
   81:
        boost::lexical_cast<int>(sn[2]),
   82:
        boost::lexical_cast<int>(sn[3]));
   83:                   tb += boost::posix_time::millisec(boost::lexical_cast<int>(s
n[4]));
   84:                   tb = tb - ta;
   85:

   86:                   temp += sp[0] + " " + sn[0] + " " + "Boot Completed";
   87:                   wtof(fn,temp);
   88:                   tb.total_milliseconds()
   89:                   wtof(fn, boottime + std::to_string(tb.total_milliseconds()))
;
   90:                   completeboot = 0;
   91:                   temp = ""
   92:           }
   93:     cout<< "No match on current line."<<endl;
   94:   }
   95: }
```

```
    1: #include <iostream>
    2: #include <string>
    3: #include <boost/regex.hpp>
    4: #include <boost/date_time.hpp)
    5: using namespace std;
    6: using namespace boost;
    7:
    8: int main (){
    9:
   10:   //using namespace boost::gregorian;
   11:   //using namespace boost::posix_time;
   12:
   13:   string t = "(\\d{2}):(\\d{2}):(\\d{2})";
   14:   //(\d{2}):(\d{2}):(\d{2})
   15:   string tmm= "(\\d{2}):(\\d{2}):(\\d{2}\\.\\d{3})";
   16:   //(\d{2}):(\d{2}):(\d{2})\.(\d{3})
   17:   string gd = "(\\d{4})-(\\d{2})-(\\d{2})";
   18:
   19:   string f = ".*log.c.166.*"; // will only find line with this text in it us
e regmatch
   20:   string fs = "2014-01-26 09:55:07: (log.c.166) server started";
   21:   //use regmatch
   22:   string fsa = "oejs.AbstractConnector:Started SelectChannelConnector";
   23:   string fsb = "2014-01-26 09:58:04.362:INFO:oejs.AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080";
   24:   string s, rs;
   25:   regex e;
   26:   smatch sm;
   27:   string temp;
   28:   while( true ) {
   29:     cin >> s;
   30:     regex e(s);
   31:     //bool match = regex_match (fs,sm,e);
   32:     bool match = regex_search(fsb,sm, e);
   33:     cout<<( match? "Matched" : "Not matched") <<endl<< endl;
   34:     if (sm.size() > 0) {
   35:       cout << "the matches were: "<<endl;
   36:       for (unsigned i=0; i<sm.size(); ++i) {
   37:         cout << "[" << sm[i] << "] " << endl;
   38:
   39:       }
   40:       /*
   41:         boost::postix_time::time_duration ta(sm[1], sm[2], sm[3]);
   42:         boost::postix_time::time_duration tb(sn[1], sn[2], sn[3]);
   43:         tb += milliseconds( (long)sn[4]);
   44:         tb = tb - ta ;
   45:         cout <<tb.total_miliseconds()<< endl;
   46:       */
   47:     }
   48:   }
   49: }
   50:
   51: /*
   52:
   53: time(){
   54: using namespace boost::gregorian;
   55: using namespace boost::posix_time;
   56:
   57: boost::postix_time::time_duration ta(sm[1],sm[2],sm[3])
   58: boost::postix_time::time_duration tb(sn[1], sn[2], sn[3])
   59: tb = milliseconds( (long)sn[4])
```

```
   60:
   61: ta = ta - tb ;
   62:
   63: ta.total_miliseconds();
   64: }
   65: vector< strings > names
   66: names.push_back("Logging");
   67: names.push_back("DatabaseInitialize");
   68: names.push_back("MessagingService");
   69: names.push_back("HealthMonitorService");
   70: names.push_back("Persistence");
   71: names.push_back("ConfigurationService");
   72: names.push_back("LandingPadService");
   73: names.push_back("PortConfigurationService");
   74: names.push_back("CacheService");
   75: names.push_back("ThemingService");
   76: names.push_back("StagingService");
   77: names.push_back("DeviceIOService");
   78: names.push_back("BellService");
   79: names.push_back("GateService");
   80: names.push_back("ReaderDataService");
   81: names.push_back("BiometricService");
   82: names.push_back("OfflineSmartviewService");
   83: names.push_back("AVFeedbackService");
   84: names.push_back("DatabaseThreads");
   85: names.push_back("SoftLoadService");
   86: names.push_back("WATCHDOG");
   87: names.push_back("ProtocolService");
   88: names.push_back("DiagnosticsService");
   89:
   90:
   91:
   92:
   93: */
```

```
    1: #include <iostream>
    2: #include <string>
    3: #include <boost/regex.hpp>
    4: #include <boost/date_time.hpp>
    5:
    6: using namespace std;
    7: using namespace boost;
    8:
    9: int main (){
   10:
   11:   //using namespace boost::gregorian;
   12:   //using namespace boost::posix_time;
   13:
   14:   string t = "(\\d{2}):(\\d{2}):(\\d{2})";
   15:   //(\d{2}):(\d{2}):(\d{2})
   16:   string tmm= "(\\d{2}):(\\d{2}):(\\d{2})\\.(\\d{3})";
   17:   //(\d{2}):(\d{2}):(\d{2})\.(\d{3})
   18:   string gd = "(\\d{4})-(\\d{2})-(\\d{2})";
   19:
   20:   //string ft = ".*log.c.166.*"; // will only find line with this text in it
 use regmatch
   21:   string fs = "2014-01-26 09:55:07: (log.c.166) server started";
   22:   //use regmatch
   23:   string fsa = "oejs.AbstractConnector:Started SelectChannelConnector";
   24:   string fsb = "2014-01-26 09:58:04.362:INFO:oejs.AbstractConnector:Started
SelectChannelConnector@0.0.0.0:9080";
   25:   string s, rs;
   26:
   27:   smatch sm, sn, so;
   28:
   29:   regex e(t);
   30:   regex f(tmm);
   31:   regex getdate(gd);
   32:   //bool match = regex_match (fs,sm,e);
   33:   bool match = regex_search(fs,sm, e);
   34:   bool matcha = regex_search(fsb,sn, f);
   35:   bool gooddate = regex_search(fs, so, getdate);
   36:
   37:   if(gooddate)
   38:     cout << so[0] << endl;
   39:   cout<<( match? "Matched" : "Not matched") <<endl<< endl;
   40:   cout<<( matcha? "Matched A" : "Not matched A") <<endl<< endl;
   41:
   42:   if (match && matcha) {
   43:
   44:     int ha,hb,hc,hd, he, hf, hg;
   45:     string example ;
   46:
   47:     example = so[0] + " " + sm[0] + " Boot Start \n";
   48:
   49:     cout << example ;
   50:     ha = boost::lexical_cast<int>(sm[1]);
   51:     hb = boost::lexical_cast<int>(sm[2]);
   52:     hc = boost::lexical_cast<int>(sm[3]);
   53:     hd = boost::lexical_cast<int>(sn[1]);
   54:     he = boost::lexical_cast<int>(sn[2]);
   55:     hf = boost::lexical_cast<int>(sn[3]);
   56:     hg = boost::lexical_cast<int>(sn[4]);
   57:     //ha = atoi( h.c_str());
   58:     boost::posix_time::time_duration ta(ha, hb, hc);
   59:     boost::posix_time::time_duration tb(hd, he, hf);
```

```
   60:     tb += boost::posix_time::millisec(hg);
   61:     tb = tb - ta ;
   62:     cout <<tb.total_milliseconds()<< endl;
   63:   }
   64: }
```

```
    1: // compile with
    2: // g++ stdin_boost.cpp -lboost_regex
    3:
    4: // regex_match example
    5: #include <iostream>
    6: #include <string>
    7: #include <boost/regex.hpp>
    8:
    9: using namespace std;
   10: using namespace boost;
   11:
   12: int main ()
   13: {
   14:
   15:
   16:   string s, rs;
   17:   regex e;
   18:
   19:   // see http://www.boost.org/doc/libs/1_55_0/boost/regex/v4/error_type.hpp
   20:   cout << "Here are some helpful error codes you may encounter\n";
   21:   cout << "while constructing your regex\n";
   22:   cout << "error_bad_pattern " << regex_constants::error_bad_pattern << endl
;
   23:   cout << "error_collate " << regex_constants::error_collate << endl;
   24:   cout << "error_ctype " << regex_constants::error_ctype << endl;
   25:   cout << "error_escape " << regex_constants::error_escape << endl;
   26:   cout << "error_backref " << regex_constants::error_backref << endl;
   27:   cout << "error_brack " << regex_constants::error_brack << endl;
   28:   cout << "error_paren " << regex_constants::error_paren << endl;
   29:   cout << "error_brace " << regex_constants::error_brace << endl;
   30:   cout << "error_badbrace " << regex_constants::error_badbrace << endl;
   31:
   32:   cout << endl;
   33:
   34:   cout << "Enter regex > ";
   35:   getline (cin, rs);
   36:
   37:   try {
   38:     e = regex (rs);
   39:   } catch (regex_error& exc) {
   40:     cout << "Regex constructor failed with code " << exc.code() << endl;
   41:     exit(1);
   42:   }
   43:
   44:   cout << "Enter line > ";
   45:
   46:   while (getline(cin, s)) {
   47:
   48:     cout << endl;
   49:
   50:     if (regex_match (s,e))
   51:       cout << "string object \"" << s << "\" matched\n\n";
   52:
   53:     if ( regex_match ( s.begin(), s.end(), e ) )
   54:       cout << "range on \"" << s << "\" matched\n\n";
   55:
   56:     smatch sm;    // same as match_results<string::const_iterator> sm;
   57:     regex_match (s,sm,e);
   58:     cout << "string object \"" << s << "\" with " << sm.size() << " matches\
n\n";
   59:     // uses constant iterators so requires -std=gnu++0x
```

```
   60:     //    regex_match ( s.cbegin(), s.cend(), sm, e);
   61:     //    cout << "range on \"" << s << "\" with " << sm.size() << " matches
\n";
   62:
   63:     if (sm.size() > 0) {
   64:       cout << "the matches were: ";
   65:       for (unsigned i=0; i<sm.size(); ++i) {
   66:         cout << "[" << sm[i] << "] " << endl;
   67:       }
   68:     }
   69:
   70:     cout << endl << endl;
   71:
   72:     cout << "Enter line > ";
   73:   }
   74:
   75:
   76:   return 0;
   77: }
```