EGRE 347 Applied Object Oriented Programming Homework #5 – State Machines – NMEA GPS sentence parsing

This homework must be your own (individual) work as defined in the course syllabus and discussed in class.

For this homework, you will be developing a program that will parse (i.e. "interpret") information that is output by a standard Global Positioning System (GPS) receiver.

Most off-the-shelf GPS receivers output their position and status information in a set of messages (called "sentences") that conform to a standard specification. This specification was first developed by the National Marine Electronics Association (NMEA) and it has since become the universal standard for GPS units.

NMEA GPS sentences consist of data in ASCII form – that is, each byte of information that is output corresponds to an ASCII character. Thus NMEA sentences are typically transmitted over an RS-232 serial link and thus are received one byte at a time. For the purposes of this homework, we will be reading the NMEA GPS sentences from a file, one byte at a time. *This is critical – your NMEA GPS sentence parser MUST handle each byte from the file separately – you can NOT read the information from the file line-by-line and deal with it that way.* Any solution turned in that does not read the file and handle the information one byte at a time will receive the maximum grade of 50% !

A valid NMEA GPS sentence starts with a '$' character. After the '$', there is a 5-character sequence starting with "GP" and ending with three characters that specify the "type" of the message. For this homework assignment, there are 4 valid message types: GPGGA, GPGSA, GPGSV, and GPRMC. Any message that does not start with one of these four message types should be ignored.

After the '$' character and message type, the sentence includes a variable number of characters that contain the actual message data. For this homework assignment, you will simply save these characters into a buffer. This buffer should be implemented as either an array of characters or a C++ string. The next homework assignment will work on interpreting the actual information in the message buffer and displaying it on the screen.

After the message data, the sentence will contain a '*' character signifying the end of the data. Following the '*' character, there will be two ASCII characters which specify the checksum of the message in hexadecimal format. After the two checksum characters, the next character should be a '$' character signifying the start of the next message. More information and examples of the NMEA GPS sentences can be found at this website (http://aprs.gids.nl/nmea/).

The checksum for the NMEA sentences is calculated over every character in the message except the '$' character at the start of the message, and the '*' and two checksum characters at the end of the message. The checksum is calculated by "exclusive OR'ing" the characters in the message, one-by-one, from the beginning to end.

Your program must calculate the checksum from the message data and compare it to the checksum transmitted with the message to determine if the message is correct. Any message in which the calculated checksum does not match the received checksum should be ignored. More information on calculating the checksum for an NMEA sentence can be found at this website: (http://rietman.wordpress.com/2008/09/25/how-to-calculate-the-nmea-checksum/)

For each correct NMEA GPS sentence that is found in the file, your program must print out the following:

Message type - either GPGGA, GPGSA, GPGSV, or GPRMC
Message data - all of the characters of the message data, starting right after the '$' character and up to, but not including the '*' character and the two checksum characters.
Message checksum - the two character message checksum in hexadecimal format

Given an input file with the following messages in it:

```
$GPGGA,155002.000,3732.7239,N,07726.9956,W,1,05,1.7,89.1,M,-33.6,M,,0000*58
$GPGSA,A,3,17,06,28,02,24,,,,,,,,3.2,1.7,2.7*3E
$GPRMC,155002.000,A,3732.7239,N,07726.9956,W,0.00,80.30,141014,,,A*48
$GPGGA,155003.000,3732.7239,N,07726.9956,W,1,05,1.7,89.1,M,-33.6,M,,0000*59
```

Your program should produce the following output:

```
> prog1 gps1.log

Message type:GPGGA
Message data:GPGGA,155002.000,3732.7239,N,07726.9956,W,1,05,1.7,89.1,M,-
33.6,M,,0000
Message checksum:58

Message type:GPGSA
Message data:GPGSA,A,3,17,06,28,02,24,,,,,,,,3.2,1.7,2.7
Message checksum:3E

Message type:GPRMC
Message data:GPRMC,155002.000,A,3732.7239,N,07726.9956,W,0.00,80.30,141014,,,A
Message checksum:48

Message type:GPGGA
Message data:GPGGA,155003.000,3732.7239,N,07726.9956,W,1,05,1.7,89.1,M,-
33.6,M,,0000
Message checksum:59
```

The following NMEA GPS sentences have two errors in them. The second sentence has corrupted data and does not have a checksum. The fourth sentence has an incorrect checksum:

```
$GPGGA,155002.000,3732.7239,N,07726.9956,W,1,05,1.7,89.1,M,-33.6,M,,0000*58
$GPGSA,A,jdksjduwls02,24,,,,,,,,3.2,1.7,2.7
$GPRMC,155002.000,A,3732.7239,N,07726.9956,W,0.00,80.30,141014,,,A*48
$GPGGA,155003.000,3732.7239,N,07726.9956,W,1,05,1.7,89.1,M,-33.6,M,,0000*58
$GPGSA,A,3,17,06,28,02,24,,,,,,,,3.2,1.7,2.7*3E
$GPGSV,3,1,12,17,64,038,42,06,63,250,43,28,44,150,45,02,25,239,38*7B
```

For this input file, your program should produce the following output:

```
> prog1 gps2.log

Message type:GPGGA
Message data:GPGGA,155002.000,3732.7239,N,07726.9956,W,1,05,1.7,89.1,M,-
33.6,M,,0000
Message checksum:58

Message type:GPRMC
Message data:GPRMC,155002.000,A,3732.7239,N,07726.9956,W,0.00,80.30,141014,,,A
Message checksum:48

Message type:GPGSA
Message data:GPGSA,A,3,17,06,28,02,24,,,,,,,,3.2,1.7,2.7
Message checksum:3E

Message type:GPGSV
Message data:GPGSV,3,1,12,17,64,038,42,06,63,250,43,28,44,150,45,02,25,239,38
Message checksum:7B
```

There is also a third file containing GPS sentences (gps3.log) that you should test your program on. This file contains a total of 66 GPS sentences which are all correct.

For this assignment, you must turn in a git bundle file (hw5.bundle) with all of your source files and a working Makefile to compile your solution. The Makefile should compile the solution in C++ with no errors or warnings. Turn in your assignment by attaching the git bundle file to the assignment submission page.

Remember the class policy on late submissions – no late submissions are allowed unless prior arrangement is made with the instructor.