TP traitement données GPS

L'objectif de ce TP est d'utiliser les notions toutes les notions abordées jusqu'à maintenant :
- boucles
- test conditionnel
- structure
- pointeur
- chaîne de caractères

Pour ce faire, l'objectif de ce TP est de construire un certains nombre d'outils sur les chaînes de caractères et sur les données GPS.

L'ensemble des exercices peut être traité à plusieurs. Précisez dans GPS_TOOLS.h les noms des auteurs (et essayez de pas faire un groupe de plus de 6 personnes). Pensez à utiliser Git et faites des commit réguliers pas forcément que quand ça marche.

Description du code fourni :

Rien !!!

Conseils :

1/ Documentez votre code ;
2/ Utilisez la fonction « main » de votre programme pour tester les différentes fonctions réalisées ;
3/ KISS (« Keep It Stupidly Simple »)
4/ On optimise jamais un code qui ne fonctionne pas !

Préambule :

La norme NMEA0183 est une norme définissant un protocole de communication pour les données de navigation. L'une des utilisations les plus commune est la transmission des données GPS.

Quelques liens utiles pour aller plus loin et à regarder avant de commencer :

https://fr.wikipedia.org/wiki/NMEA_0183

https://gpsd.gitlab.io/gpsd/NMEA.html

Nous n'allons pas traiter l'ensemble du protocole et nous concentrer sur la GGA. Libre à vous ensuite d'ajouter les autres trames.

Sauf mention contraire, l'ensemble des fonctions qui seront réalisées dans la suite du TP doivent être déclarées dans GPS_TOOLS.h et définies dans GPS_TOOLS.c. La fonction main est définie dans le fichier main.c

Exercice 1 :

Dans le fichiers GPS_TOOLS.h, déclarer un énuméré GPS_FRAME_TYPE contenant les symbôles suivants : UNKNOWN_FRAME=0,XRMC_FRAME,XGGA_FRAME, XGLL_FRAME, XGNS_FRAME

Exercice 2 :

Définissez dans GPS_TOOLS.h la structure GPS_DATA qui contient les champs suivants :
- 1 pointeur vers char nommé « frame »
- 1 entier nommé « frame_type» ;
- 1 entier nommé « is_valid » ;
- 5 entiers nommés respectivement « year », « month », « day », « hours », « minutes » ;
- 1 double nommé « seconde » ;
- 1 double nommé « latitude » ;
- 1 double nommé «longitude» ;
- 1 double nommé « altitude ».

Définissez le type gps_data correspondant à la structure précédemment définie.

Exercice 3 :

Déclarez et définissez la fonction

char* findChar(const char *str, char c, int pos)

Cette fonction permet de rechercher la première occurrence du caractère « c » dans la chaîne de caractères « str » à partir du « pos»ième caractères . Si le caractère « c » est trouvé, la fonction retourne le pointeur vers le caractère dans la chaîne « str ». Sinon la fonction renvoie la valeur NULL.

Exercice 4 :

Déclarez et définissez la fonction

char* findStr(const char *str, const char *s, int pos)

Cette fonction permet de rechercher la première occurrence de la chaîne de caractères « s » dans la chaîne de caractères « str » à partir du « pos»ième caractères . Si la chaîne « s » est trouvée, la fonction retourne le pointeur vers le premier caractère de la chaîne trouvée. Sinon la fonction renvoie la valeur NULL.

Exercice 5 :

Déclarez et définissez la fonction

int isValidHeader(const char *frame)

Cette fonction recherche dans la chaîne de caractère « frame » un entête valide de trame NMEA. Si un entête d'un trame GGA est trouvé, elle retourne la valeur XGGA_FRAME. Si l'entête trouvé est de type RMC, elle retourne la valeur XRMC_FRAME, … Si aucun entête valide est trouvé, elle retourne la valeur UNKNOWN_FRAME.

Dans un premier temps, vous pouvez traiter uniquement le cas de la trame GGA. Pensez à utiliser les fonctions précédemment écrites.
A noter qu'un entête est défini de la manière suivante « $xyGGA, » où xy sont 2 caractères pouvant prendre n'importe quelle valeur entre 'A' et 'Z'.

Exercice 6 :

Déclarez et définissez la fonction

int getFieldsCount(const char *frame)

Cette fonction compte le nombre de champs que contient la trame NMEA. Les champs d'une trame NMEA sont séparés par une « , » et sont compris entre le « $ » de début de trame et le « * » indiquant le check sum. L'entête de trame est donc le premier champ.

Exercice 7 :

Déclarez et définissez la fonction

char *gotoField(const char* frame, int n)

Cette fonction renvoie le pointeur vers le premier caractère du « n »ieme champs de la trame contenu dans frame. Si le « n »ieme champ n'est pas trouvé, la fonction retour NULL.

Exercice 8 :

Déclarez et définissez la fonction

char hexToChar(unsigned char value)

Cette fonction renvoie le caractère ASCII majuscule correspondant à la valeur hexadécimale de « value ». « value » doit avoir une valeur entre 0 et 15 sinon le résultat n'est pas garanti.
Exemples :

pour value =1 la fonction retourne '1'
pour value =12 la fonction retourne 'C'

Exercice 9 :

Déclarez et définissez la fonction

int computeCheckSum(const char *frame)

L'objectif de cette fonction est de calculer la valeur numérique du checksum de la trame. Le checksum d'un trame NMEA est égale à un XOR (opérateur ^ en C) entre tous les caractères de la

trame qui sont compris entre le ‘$’ de début de trame et le ‘*’ de délimitation de checksum.
Exemple :
la trame $GPGGA,064036.289,4836.5375,N,00740.9373,E,1,04,3.2,200.2,M,,,,0000*0E
a un checksum égale à 15.

Exercice 10 :

Déclarez et définissez la fonction

int isValidGpsData(const char *str_signal)

Cette fonction analyse si la chaîne de caractère « str_signal » contient une trame NMEA valide. Si ce n’est pas le cas elle retourne la valeur UNKNOWN_FRAME. Dans le cas d’une trame valide elle renvoie le valeur énumérée correspondant à la trame (XRMC_FRAME,XGGA_FRAME, …)

Une trame NMEA est considérée valide si :
- Son entête est correct ;
- Elle contient au maximum 82 caractères
- Elle contient un nombre de champs valables (attention certaines trames peuvent avoir un nombre de champs non fixe, par exemple la RMC peut avoir 12, 13 ou 14 champs en plus de l’entête).
- Elle a un checksum valide.

Exercice 11 :

Déclarez et définissez la fonction

int extractGpsData(const char *frame, gps_data *data)

Cette fonction permet de valider si une trame NMEA est contenue dans « frame » et de remplir la structure « data » dont le pointeur est passé en paramètre avec les informations contenues dans la trame NMEA « frame ». Cette fonction renvoie l’énuméré de type de trame à l’instar de la fonction **isValidGpsData.**

Notez bien que les données de latitude et de longitude devront être convertie en degrés (en non degrés/minutes) pour être utilisées dans la structure. Les latitude Nord sont positives et les Sud sont négatives. De la même manière les longitude Est sont positives et le Ouest sont négatives.

Exercice 12 :

Déclarez et définissez la fonction

void printGpsData(const gps_data *data)

Cette fonction permet d’afficher l’ensemble des information contenue dans la structure « data ».

Pour aller encore plus loin :

Ça ne vous suffit pas ? vous en voulez encore ?

Voici quelques pistes d’exercices supplémentaires :

- calculez la distance entre 2 points GPS en mètre ;

- calculez la vitesse entre 2 points GPS en mètre par seconde ;

- calculez le cap entre 2 points GPS en degrés ;

Notez que les méthode de calcul doivent tenir compte de la norme WGS84 pour la rotondité de la terre.