

1 Objectifs

Le but de ce TP est de manipuler par programmation, une trame GPS de type NMEA afin d'en extraire l'information de position et l'heure. Cette trame est transmise sur une liaison série sous forme d'une chaîne de caractères. Ce sera l'occasion idéale d'utiliser les fonctions permettant de manipuler les chaînes de caractères en langage C avec des pointeurs.

2 Le capteur GNSS

Un capteur ou puce GNSS est capable de se géolocaliser grâce à la réception de signaux émis par des satellites artificiels dont les positions dans l'espace sont connues avec précision. En combinant la mesure simultanée de la distance d'au moins quatre satellites, le récepteur est capable par **multilatération** de fournir la position et l'altitude avec une précision de l'ordre du décamètre, la vitesse avec une précision de quelques m/s et le temps avec une précision atomique (ms). Un capteur fixe au sol permet, après une intégration sur une période de plusieurs minutes, de connaître la position d'un point avec une précision centimétrique.

La méthode de trilatération permet théoriquement de calculer position, vitesse et temps en utilisant le signal de trois satellites : la distance à laquelle se situe un satellite positionne l'utilisateur à la surface d'une sphère dont le centre est le satellite. L'intersection de 3 sphères permet d'identifier un point unique dans l'espace. Un quatrième satellite au minimum est néanmoins requis pour permettre de déterminer le décalage des horloges et réduire les incertitudes liées aux autres sources de perturbation du signal, on parle alors de multilatération.

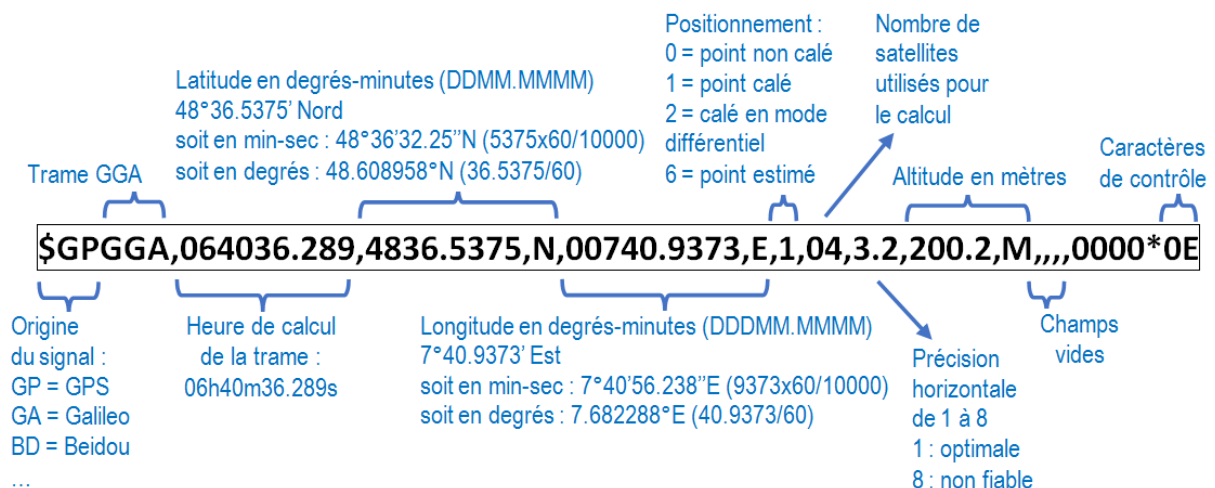
Le capteur génère une trame (une ligne de texte) regroupant plusieurs informations comme l'heure, la latitude, la longitude, l'altitude, etc.

Afin que les équipements GPS puissent tous être compatibles, il faut que cette transmission ait toujours la même forme ! Ainsi, la National Marine Electronics Association a créé un standard de communication la **NMEA 0183**.

Sous ce standard, toutes les données sont transmises sous la forme de caractères **ASCII**, tous imprimables, ainsi que les caractères [CR] Retour Chariot et [LF] Retour à la ligne, à la vitesse de transmission de **4800** bauds. Les données sont transmises sous forme de trames (phrases), dont la plus utilisée aujourd'hui s'appelle la trame **GGA**,

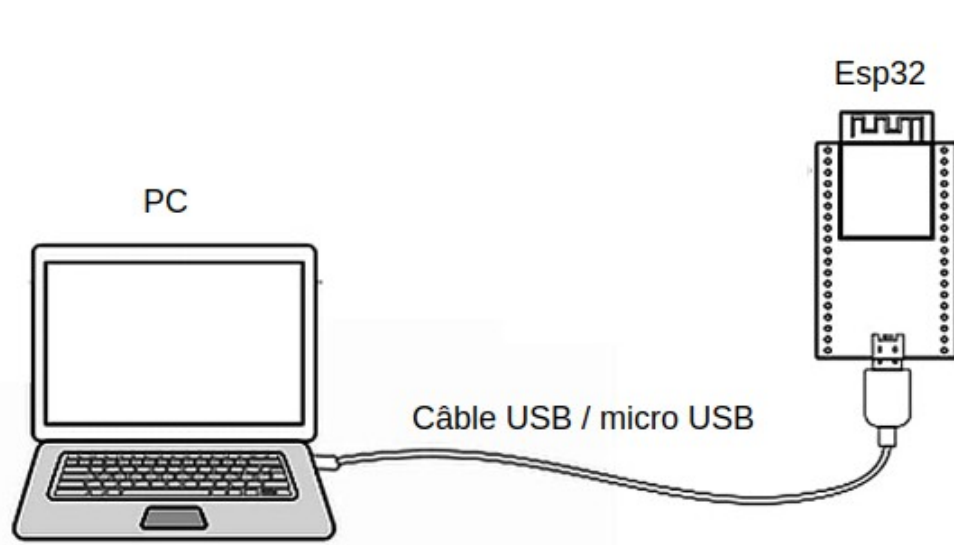
Chaque trame commence par le caractère \$. Suivent ensuite un certain nombre de champs séparés par une "virgule". Le rôle de la virgule est d'être le séparateur de champs, qui permet la dé-concaténation des données dans le programme de traitement des données.

Voici un exemple pour comprendre à quoi ressemble cette trame. On remarque que chaque donnée est séparée par une virgule même si le champ est vide et que les valeurs décimales utilisent le point :



3 Mise en œuvre du capteur GNSS simulé avec l'Esp32

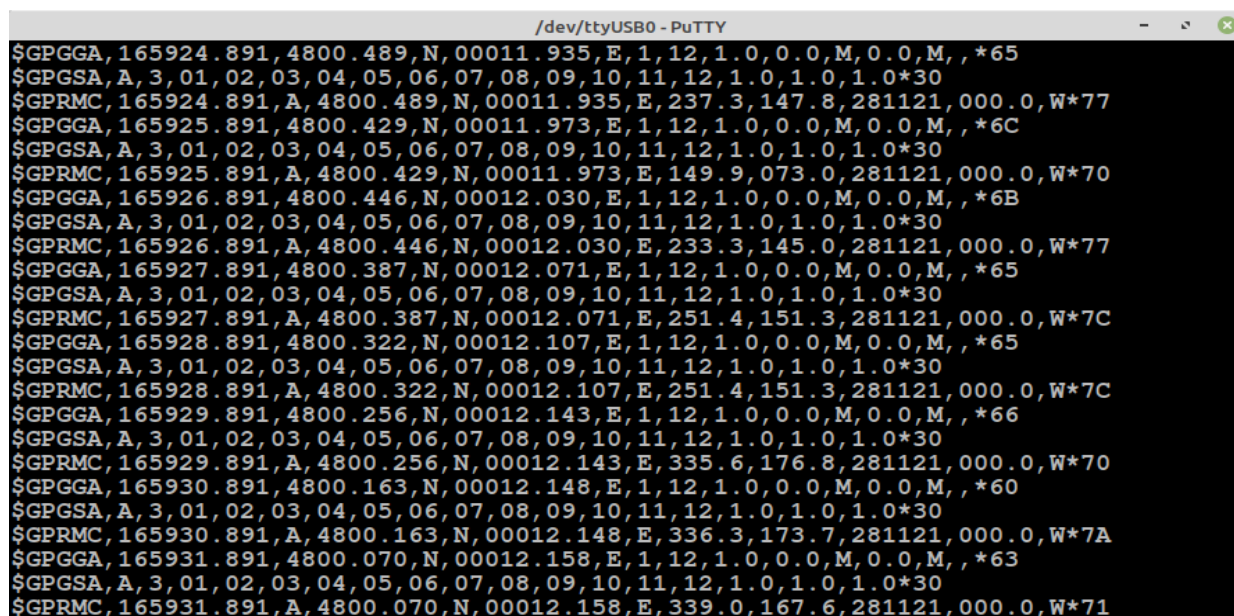
En classe , il est peu pratique de mettre en œuvre un capteur GNSS réel car l'antenne du récepteur doit être placée près d'une fenêtre pour être en mesure de capter les signaux émis par les satellites. Aussi pour simplifier les manipulation nous utiliserons un ESP32 avec un programme de simulation NMEA. Le programme émet en continu des trames NMEA sur la liaison série.



4 Visualisation des trames émises avec Putty

Configurer Putty en **Serial** avec **4800** bauds **8** bits par caractère **1** bit de stop, pas de parité pas de contrôle de flux.

Envoyer le caractère **n** pour démarrer l'envoi des trames NMEA. Les trames NMEA sont envoyés en continu.



```
/dev/ttyUSB0 - PuTTY
$GPGGA,165924.891,4800.489,N,00011.935,E,1,12,1.0,0.0,M,0.0,M,,*65
$GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GPRMC,165924.891,A,4800.489,N,00011.935,E,237.3,147.8,281121,000.0,W*77
$GPWCH,165925.891,4800.429,N,00011.973,E,1,12,1.0,0.0,M,0.0,M,,*6C
$GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GPRMC,165925.891,A,4800.429,N,00011.973,E,149.9,073.0,281121,000.0,W*70
$GPGGA,165926.891,4800.446,N,00012.030,E,1,12,1.0,0.0,M,0.0,M,,*6B
$GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GPRMC,165926.891,A,4800.446,N,00012.030,E,233.3,145.0,281121,000.0,W*77
$GPWCH,165927.891,4800.387,N,00012.071,E,1,12,1.0,0.0,M,0.0,M,,*65
$GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GPRMC,165927.891,A,4800.387,N,00012.071,E,251.4,151.3,281121,000.0,W*7C
$GPWCH,165928.891,4800.322,N,00012.107,E,1,12,1.0,0.0,M,0.0,M,,*65
$GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GPRMC,165928.891,A,4800.322,N,00012.107,E,251.4,151.3,281121,000.0,W*7C
$GPWCH,165929.891,4800.256,N,00012.143,E,1,12,1.0,0.0,M,0.0,M,,*66
$GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GPRMC,165929.891,A,4800.256,N,00012.143,E,335.6,176.8,281121,000.0,W*70
$GPWCH,165930.891,4800.163,N,00012.148,E,1,12,1.0,0.0,M,0.0,M,,*60
$GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GPRMC,165930.891,A,4800.163,N,00012.148,E,336.3,173.7,281121,000.0,W*7A
$GPWCH,165931.891,4800.070,N,00012.158,E,1,12,1.0,0.0,M,0.0,M,,*63
$GPGSA,A,3,01,02,03,04,05,06,07,08,09,10,11,12,1.0,1.0,1.0*30
$GPRMC,165931.891,A,4800.070,N,00012.158,E,339.0,167.6,281121,000.0,W*71
```

Il existe plusieurs trames correspondant à des besoins différents. Chaque trame possède une syntaxe différente. Nous nous intéresserons à la trame la plus utilisée pour connaître la position courante du récepteur : La trame **GGA**.

Une trame est constituée de **champs**. Les champs **sont séparés entre eux par des virgules**. Un champ peut être vide mais **la présence de la virgule est obligatoire**.

Nom du champ	Exemple de valeur	Signification
Type de trame	\$GPGGA	Indique qu'il s'agit d'une trame \$GP (pour GPS) de type GGA
Heure	064036.289	Signifie que la trame a été envoyée à 6h40min36.289 s
Latitude	4836.5375	Latitude en degré, minute décimale 48° 36,5375'
Indicateur Latitude	N	N : Nord , S : Sud
Longitude	00740.9373	Longitude en degré, minute décimale 007° 40,9373'
Indicateur longitude	E	E : Est , W :Ouest
Positionnement	1	0 = point non calé, 1 = point calé, 2 = point calé en mode différentiel, 6 point estimé
Nb de satellites	4	Nombre de satellites utilisés pour le calcul. La précision du positionnement dépend du nombre de satellites détectés
Précision	1.0	Dilution horizontale de la précision. Permet de connaître la fiabilité du calcul. 1 : Valeur optimale, 2 à 3 : excellente, 5 à 6 : bonne, supérieure à 8 : Mesure non fiable
Altitude	200	Altitude de l'antenne par rapport au niveau de la mer
Unité altitude	M	Unité d'altitude
Somme de Contrôle	*68	Somme de contrôle

La somme de contrôle à la fin de chaque phrase est le OU EXCLUSIF (XOR) de tous les octets de la phrase à l'exclusion du premier caractère (\$) et jusqu'au caractère avant l'étoile (*)

5 Programmation en langage C

5.1 Fonctionnalité du programme

Le programme à réaliser aura les fonctionnalités suivantes :

1. **Découpage de la trame** : Chaque champ de la trame pourra être extrait en fonction de sa position dans la trame. L'index 0 représente le champ n°1 et ainsi de suite
2. **Contrôle de la validité** la somme de contrôle calculée doit être la même que celle reçue.
3. **Affichage de la longitude, la latitude** : La longitude et la latitude devront être affichés sous forme degré décimaux.
4. **Affichage de l'heure d'envoi de la trame** : L'heure devra être affichée sous forme hh:mm:ss

5.2 Principe du décodage de trame

le principe du décodage consiste à découper la trame suivant la virgule pour la transformer en un tableau de chaîne de caractères, chaque élément du tableau correspond à un champ de la trame.

Le prototype de la fonction sera le suivant:

```
int decouper(char trame[], char *champs[], int size);
```

Algorithme proposé

Fonction **Découper ()**

paramètre d'entrée : **trame** une chaîne de caractères

paramètre de sortie : **champs** un tableau de chaîne de caractères

paramètre d'entrée : **size** entier la taille maxi du tableau

Schéma algorithmique

début

```
ptr ← @trame[0]
fin ← faux
i ← 0
```

faire

```
| tant que ( *ptr == ' ' )
| | ptr ← ptr + 1
| fin tant que
|
| si ( *ptr == '\0' ) alors
| | fin ← vrai
| sinon
| | champs[i] ← ptr
| | // parcourt la trame à la recherche de la virgule suivante
| | tant que ( *ptr ≠ ',' et *ptr ≠ '\0' )
| | | ptr ← ptr + 1
| | fin tant que
| | // Remplace la virgule par '\0' et passage au champ suivant
| | si ( *ptr ≠ '\0' ) alors
| | | *ptr ← '\0'
| | | ptr ← ptr + 1
| | fin si
| |
| | i ← i + 1
| fin sinon
|
tant que ( i < (size - 1) et pas fin)
```

```
champs[i] ← NULL
retourner i
```

fin

Variables locales

ptr adresse d'un caractère de la trame

fin drapeau indiquant que la fin de trame est atteinte.

i entier contient le nombre de champ trouvé dans la trame

1. Coder en langage C la fonction **découper**

2. Proposer un algorithme pour calculer la somme de contrôle.
coder votre l'algorithme dans une fonction pour qu'elle retourne la
somme de contrôle sous forme de chaîne de caractères en
hexadécimal.

```
/**
 * @brief Fonction pour calculer le champ CRC d'une trame NMEA
 * @param une chaîne de caractère trame
 * @return une chaîne de caractère CRC
 */
void calculerCRC(char trame[], char retour[]) {

    char *s; // pointeur pour parcourir la trame
    // Calcul le CRC entre $ et *
    char crc = 0;


    snprintf(retour, 4, "%02X", crc);
}
```

On donne le code d'une fonction pour extraire une sous chaîne d'une chaîne

```
/**
 * @brief Retourne un segment de chaîne
 * @param src      un pointeur sur la chaîne source
 * @param offset   la chaîne retournée commencera au caractère numéro offset
 * @param len      la chaîne retournée contiendra length caractères
 * @return la fonction renvoie un pointeur sur la chaîne résultat dest.
 */
char *substr(char *src, const int offset, const int length) {
    char *dest = NULL;
    if (length > 0) {
        // allocation de mémoire avec mise à zéro
        dest = calloc(length + 1, sizeof (char));
        // vérification de la réussite de l'allocation
        if (NULL != dest) {
            strncat(dest, src + offset, length);
        }
    }
    return dest;
}
```

3. Coder une fonction pour convertir les angles de degré minute en degré décimaux.

Le prototype sera :

```
double convertirDegreDeci(char *champ, char *indicateur)
```

On utilisera la fonction atof pour convertir une chaîne en double.

les minutes d'angle, de symbole ' (une apostrophe) 1 degré = 60 minutes d'angle

les secondes d'angle de symbole " (une double apostrophe) 1 minute d'angle = 60 secondes d'angle et donc 1 degré = 60*60 = 3600 secondes d'angle.

Attention de pas oublier de libérer la mémoire, à chaque calloc doit correspondre un free.

4. Coder une fonction pour obtenir l'affichage de l'heure en hh:mm:ss