Lycée polyvalent DIDEROT **Département IRIS**
61, rue David d'Angers
75019 PARIS
http://www.diderot.org

TP

Programmation trame GPS

Manipulation de trame GPS NMEA		
Auteur	**Version - Date**	**Nom du fichier**
G.VALET	Version 1.2 - Oct 2010	tp-prog-gps.docx
Ce TP a pour but de manipuler par programmation, une trame GPS de type NMEA. Cette trame est acquise 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		

A. Sommaire

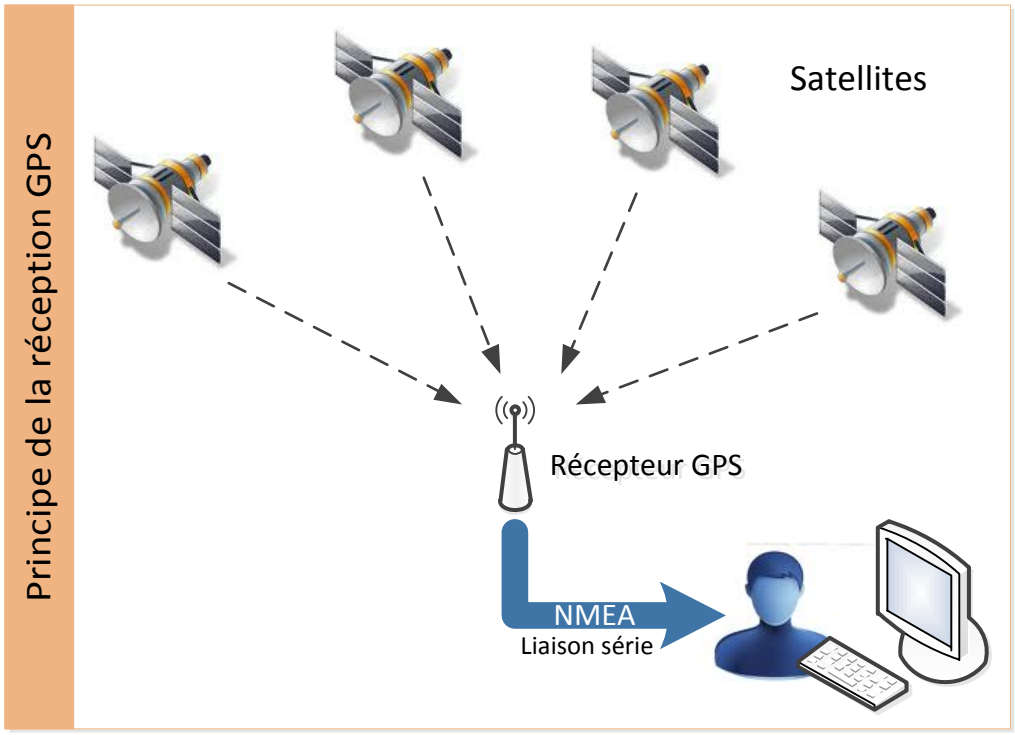
A. SOMMAIRE 2
B. OBJECTIFS 3
C. LA RECEPTION GPS 3
D. LA TRAME NMEA 3
D.1. Principe de base *3*
D.2. La trame GGA *4*
E. TRAVAIL A REALISER 5
E.1. Cahier des charges *5*
E.2. Le traitement des chaînes de caractère *5*
E.3. Principe du traitement de la trame *5*
E.4. Fonction « extraireChamp » *5*
a. Principe de l'extraction 5
b. Algorithme proposé 6
c. Codage de la fonction 6
E.5. Fonction de comptage des champs *6*
E.6. Vérification de la trame *7*
E.7. Conversion des données de localisation *7*
E.8. Affichage de l'heure *7*
E.9. Communication avec le récepteur GPS *8*
F. ANNEXE : CHAINES DE CARACTERES EN C 9
F.1. Déclarer une chaîne *9*
a. Chaîne non modifiable 9
b. Chaîne modifiable 9
F.2. Afficher une chaîne *10*
F.3. Initialiser une chaîne modifiable *10*
a. Allocation de mémoire 10
b. Allocation et initialisation de mémoire 10
F.4. Concaténation de chaîne *11*
F.5. Connaître la taille d'une chaîne *11*
F.6. Formater une chaîne *11*
G. ANNEXE : BIBLIOTHEQUE DE COMMUNICATION AVEC LE GPS 12

B. Objectifs

Ce TP a pour but de manipuler par programmation, une trame GPS de type NMEA. Cette trame est acquise 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.

C. La réception GPS

Un récepteur GPS est capable de se géolocaliser grâce à la réception de signaux émis par des satellites géostationnaires. Le récepteur GPS détermine par calcul sa position et peut la transmettre sous forme d'une trame NMEA.



Le calcul de la position est effectué par le récepteur qui fabrique **une trame de caractères NMEA**.

Cette trame est ensuite envoyée via une liaison série.

L'ordinateur décode la trame et affiche éventuellement la carte correspondant à la position GPS reçue

D. La trame NMEA

D.1. Principe de base

Une trame NMEA (*National Marine Electronics Association*) est une suite de caractères contenant des informations de géolocalisation comme :

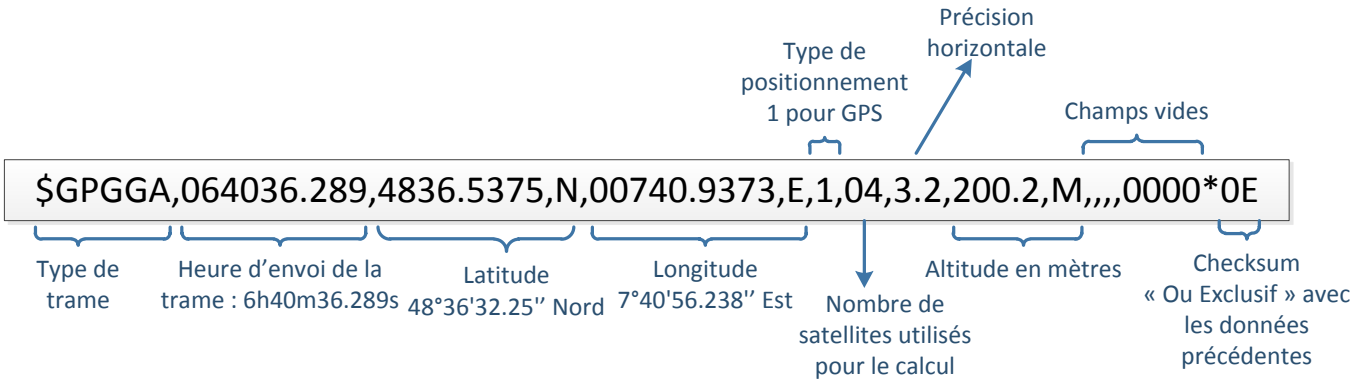
- La latitude, la longitude,
- La vitesse, l'altitude
- Le nombre de satellites
- L'heure, la date
- …

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**.

D.2. La trame GGA

Exemple de trame GGA :



On remarquera que chaque champ de la trame est séparé par des virgules. Voici un tableau récapitulant la signification des champs :

Nom du champ	Ex de valeur	Signification
Type de trame	$GPGGA	Indique qu'il s'agit d'une trame de type GGA
Heure	064036.289	Signifie que la trame a été envoyée à 6h40min36.289 s
Latitude	4836.5375	Latitude en deg, min, sec. Les secondes doivent être convertit en base 60 : 5375/100*60 = 3225 , soit 32.25 s
Indicateur Latitude	N	N : Nord , S : Sud
Longitude	00740.9373	Longitude en deg,min,sec soit 7°40'56.238''
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	04	Nombre de satellites utilisés pour le calcul. La précision du positionnement dépend du nombre de satellites détectés
Précision	3.2	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	200.2	Altitude de l'antenne par rapport au niveau de la mer
Unité altitude	M	Altitude en mètres

E. Travail à réaliser

E.1. Cahier des charges

Le programme à réaliser aura les caractéristiques suivantes :

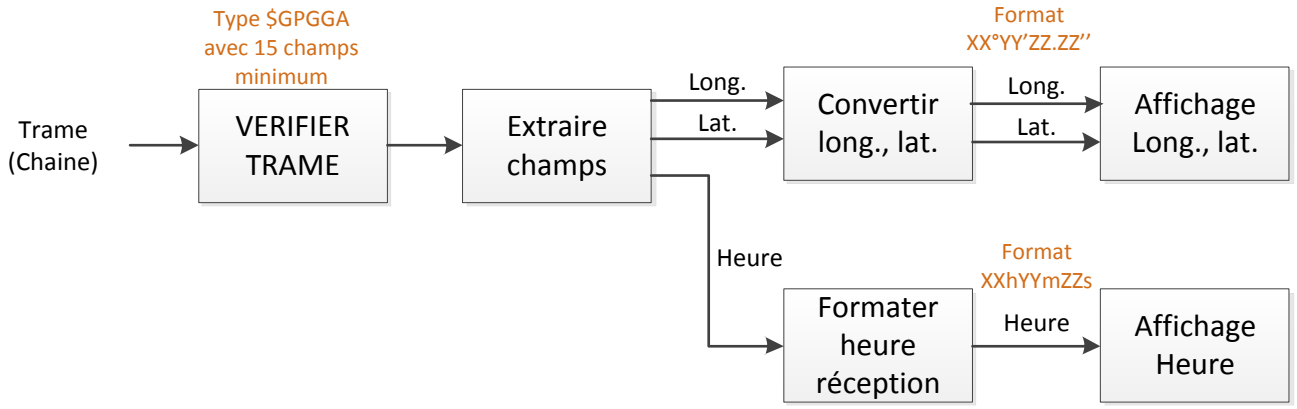
- **Décodage de la trame** : Chaque champ de la trame pourra être extrait en fonction de sa position dans la trame. L'index 0 représente le champ n°1 et ainsi de suite
- **Affichage de la longitude, la latitude** : La longitude et la latitude devront être affichés sous forme degré, minute, secondes
- **Affichage de l'heure d'envoi de la trame** : L'heure devra être affichée sous forme h,m,s

E.2. Le traitement des chaînes de caractère

Voir Annexe : Chaînes de caractère en C

E.3. Principe du traitement de la trame

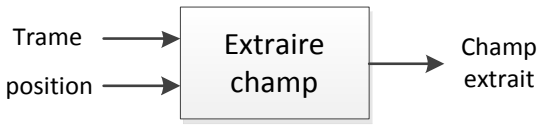
La trame sera analysée caractère par caractère afin d'en extraire les champs « longitude », « latitude » et « heure d'émission » :



E.4. Fonction « extraireChamp »

a. Principe de l'extraction

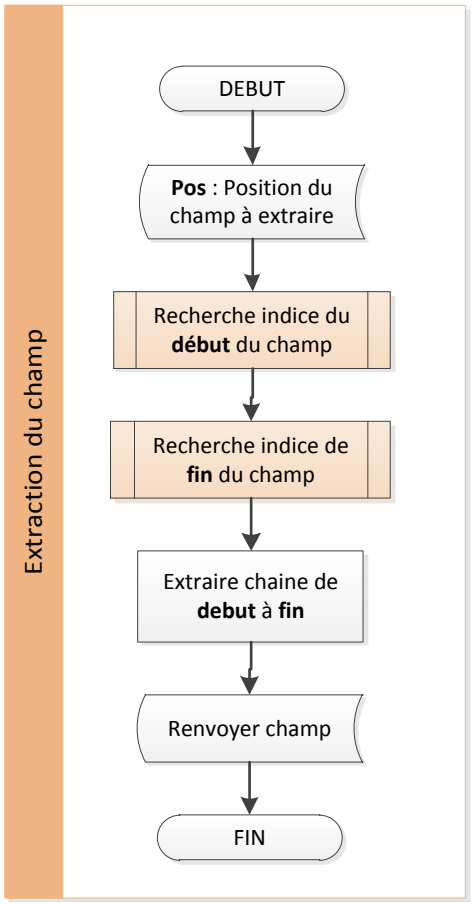
Le but est de coder une fonction capable d'extraire n'importe quel champ de la trame en fournissant simplement sa position au sein de cette trame :

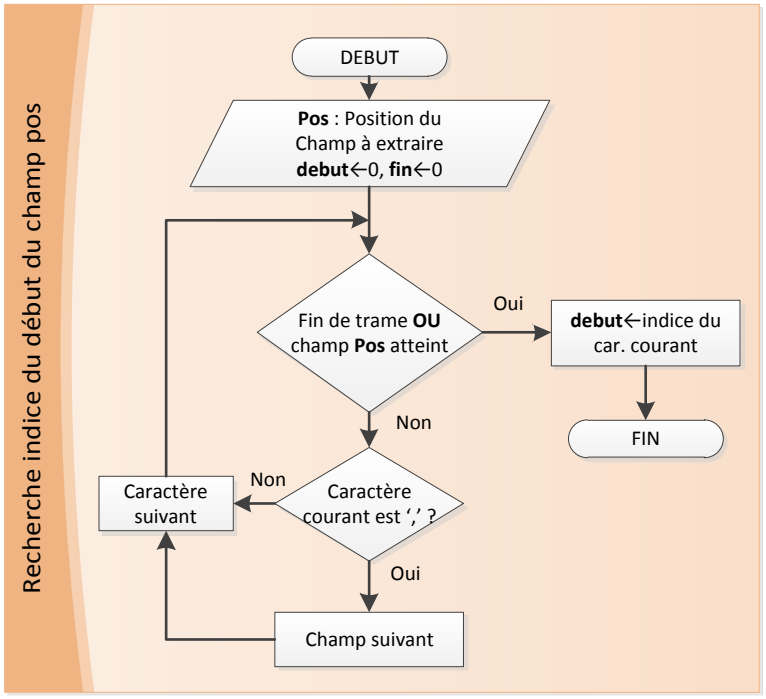


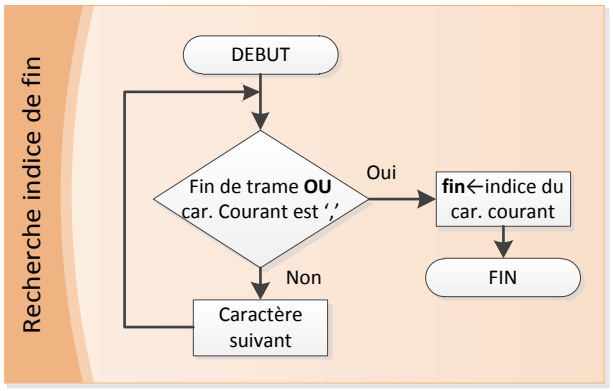
Quel sera le prototype de la fonction extraireChamp :

```
__________   extraireChamp( ______________ ) ;
```

b. Algorithme proposé







c. Codage de la fonction

Réaliser la fonction « extraireChamp » en suivant l'algorithme précédent. Testez l'extraction depuis le programme principal en testant plusieurs champs et notamment le 1er et le dernier

E.5. Fonction de comptage des champs

Réalisez une fonction capable de compter le nombre de champs de la trame. Compter le nombre de champ revient simplement à compter le nombre de virgules présentes dans la trame.

Prototype de la fonction : _____ compterChamps(_____) ;

Codez et testez le bon fonctionnement de la fonction

E.6. Vérification de la trame

L'étape de vérification consiste à contrôler le nombre de champs qu'elle contient mais également que le nom du 1er champ est bien « $GPGGA ».

Codez cette vérification au début du programme principal. Si la vérification échoue, le programme devra se terminer.

E.7. Conversion des données de localisation

L'affichage des informations de localisation doit se faire sous la forme :

```
Latitude : 7°40'56.238" E
Longitude : 48°36'32.250" N
```

Pour parvenir à ce résultat, on utilisera la fonction suivante :

```
char * convert(char* ch) {

    double f = atof(ch);

    int deg,min;
    double sec;

    deg = (int)(f / 100.0);
    min = (int)(f - (deg * 100.0));
    sec =  60.0*(f - (deg*100.0) - min);

    char *s = calloc(14,sizeof(char));

    sprintf(s,"%3d%c%2d'%5.3f",deg,0xF8,min,sec);

    return s;
}
```

Ajoutez cette fonction à votre programme et testez-la sur les champs « Latitude » et « Longitude ».

E.8. Affichage de l'heure

L'objectif est d'afficher l'heure sous forme : 15h14m44s

Créez une fonction « convheure » appropriée qui renvoie une chaîne au bon format.

E.9. Communication avec le récepteur GPS

Le récepteur GPS « MGL-30R/U » connecté au port série d'un PC permet de recevoir sur le PC les trames NMEA.

Le récepteur est connecté au PC par une liaison RS232. Il est alimenté par un port PS2 souris (Ajouter un adaptateur PS2/USB sur les ordinateurs récents)

La partie liaison série n'est pas traitée dans ce TP. Néanmoins, une bibliothèque est fournie afin de permettre la communication (Voir Annexe G)

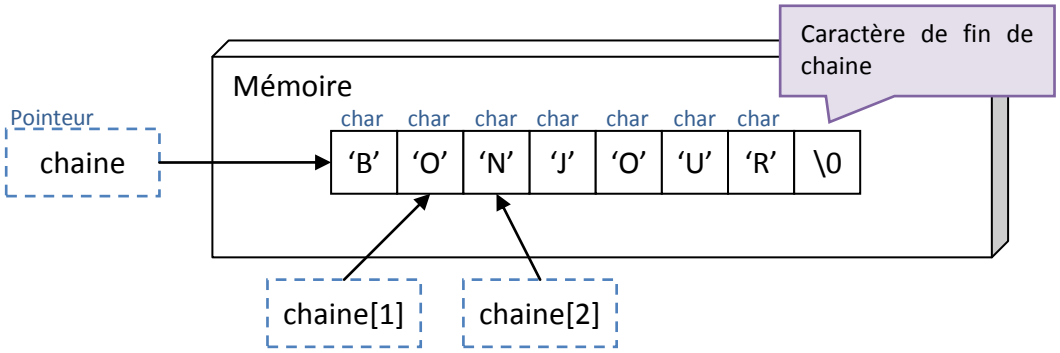
F. Annexe : Chaînes de caractères en C

Contrairement à beaucoup de langages plus récents, le type chaîne de caractère n'existe pas en langage C. Nous considérerons donc une chaîne comme un tableau de caractères de type char :

Voici comment une chaîne de caractère est traitée en C :

- Pour déclarer une chaîne, il faut utiliser le type « char * »
- Une chaine doit impérativement se terminer par un caractère de fin de chaîne (\0)
- La variable « chaine » est en fait un pointeur qui pointe sur le 1er caractère de la chaîne
- Les caractères sont stockés dans un espace contigu de la mémoire. Un caractère est codé sur 1 octet. Une chaîne de 7 caractères est codée sur 8 octets (NE pas oublier le caractère de fin de chaîne)

```
char * chaine = "BONJOUR" ;
```



F.1. Déclarer une chaîne

a. Chaîne non modifiable

Le principe est d'utiliser le type « char * » :

```
const char *machaine = "Une chaine de caractère";
```

La chaîne est alors stockée dans une zone de la mémoire NON MODIFIABLE. Voilà pourquoi il est conseillé de préciser qu'il s'agit d'une constante (Mot « const »).

b. Chaîne modifiable

Pour une chaîne modifiable, il vaut mieux utiliser la syntaxe suivante qui alloue dynamiquement la mémoire nécessaire :

```
char *unechaine = malloc(11*sizeof(char));    // 10 caractères + le "\0"

free(unechaine) ;     //Libère la mémoire à la fin du programme : INDISPENSABLE !!!
```

F.2. Afficher une chaîne

La fonction « printf » permet d'afficher une chaîne. Cette fonction affiche tous les caractères jusqu'au caractère de fin de chaîne.

Il faut faire très attention à vérifier qu'il y a toujours un caractère de fin de chaîne sinon, le comportement de « printf » peut provoquer des effets inattendus. En effet, cette fonction continue à afficher les caractères tant qu'elle ne rencontre pas le caractère de fin de chaine.

```
char *chaine1 = "Le BTS IRIS"; //Ajoute automatiquement le \0
char *chaine2 = malloc(11*sizeof(char)) ;

printf("%s", chaine1 ) ;
printf("%s" , chaine2 ) ; // ATTENTION COMPORTEMENT IMPREVISIBLE

free(chaine2);
```

F.3. Initialiser une chaîne modifiable

a. Allocation de mémoire

Dans le code précédent, l'appel de « printf() » sur « chaine2 » peut provoquer un affichage inattendu. La raison est la suivante :

- Un espace mémoire a été réservé pour « chaine2 » mais aucune initialisation n'a été faire. Cela signifie que cette zone mémoire peut déjà contenir des valeurs indéterminées.
- Il faut donc veiller à définir chaque caractère de la chaîne en n'omettant jamais le \0

```
char *chaine2 = malloc(8*sizeof(char));

chaine2 = strcpy(chaine2,"Bonjour");   // strcpy ajoute automatiquement le \0

free(chaine2) ;
```

b. Allocation et initialisation de mémoire

Il peut être très utile de réserver un espace mémoire ET d'initialiser celle-ci avec des 0 (Caractère de fin de chaîne). Dans ce cas, on utilise « calloc() » au lieu de « malloc » :

```
char *chaine2 = calloc(8, sizeof(char));

free(chaine2) ;
```

F.4. Concaténation de chaîne

Permet d'ajouter à la fin d'une chaîne, une autre chaîne. Voici un exemple :

```
char *chaine = malloc(50*sizeof(char));   // Chaîne de 50 caractères

// Initialise la chaîne avec bonjour
chaine2 = strcpy(chaine2,"Bonjour");   // strcpy ajoute automatiquement le \0

const char *chaine3 = " messieurs et mesdames" ;

// Ajoute la chaîne messieurs
chaine = strcat(chaine, chaine3); // Ajoute la chaine "chaine3" ET le \0
free(chaine);
```

Autre exemple mais avec la fonction « strncpy » pour ne copier qu'une partie des caractères :

```
char *chaine = malloc(50*sizeof(char));   // Chaîne de 50 caractères

// Initialise la chaîne avec bonjour
chaine2 = strcpy(chaine2,"Bonjour");   // strcpy ajoute automatiquement le \0

const char *chaine3 = " messieurs et mesdames" ;

// Ajoute la chaîne messieurs
chaine = strncat(chaine, chaine3,10); // Ajoute seulement « messieurs » SANS le \0
free(chaine);
```

Attention : Dans le code précédent, aucun caractère de fin de chaîne n'a été ajouté par « strncat ». Cela signifie que la chaîne n'est pas terminée. Une bonne pratique consiste à initialiser la zone mémoire qui contient la chaîne avec des 0 (Dans le cas de zone mémoire d'une taille raisonnable)

F.5. Connaître la taille d'une chaîne

La fonction « strlen » renvoie la taille d'une chaîne sans compter le \0 :

```
const char *ch = "Le langage C";   //  \0 ajouté automatiquement

int taille = strlen(ch) ; //Renvoie la valeur 12
```

F.6. Formater une chaîne

La fonction « sprintf » permet de créer des chaînes en utilisant un format particulier :

```
char *chaine = malloc(50*sizeof(char));

int h=12; int m=20; int s=36;
sprintf(chaine, "%d h %d m %d s",h,m,s); //Formate la chaîne avec « 12 h 20 m 36 s»
```

Dans cet exemple, la fonction sprintf utilise comme 2ème paramètre, une chaîne de formatage qui précise que l'on souhaite formater la chaîne avec 3 entiers séparés par les caractères « h », « m » et « s ».

G. Annexe : Bibliothèque de communication avec le GPS

Voici le fichier d'entête qui définit les fonctions nécessaires pour recevoir les trames :

```
/*
* FICHIER D'ENTETE DU MODULE DE COMMUNICATION
* AVEC LE RECEPTEUR GPS
*
*  Auteur : G.VALET
*  Version : 1.0 - Oct 2010
*/


/* Lecture d'une trame et récupération dans "trame"
 *  VALEUR RENVOYEE : Nombre d'octets lus
*/
int lectureTrameGGA(char *trame) ;

/* Ouverture du port Série "port"
*  port : Chemin du port série ( /dev/ttyS0 sur une machine Linux)
*  VALEUR RENVOYEE :  -1 si échec et 0 dans le cas contraire
*/
int openGPS(char *port) ;

/* Fermeture du port Série
*   Renvoie -1 si échec de fermeture
*/
int closeGPS();
```

Le fichier « comgps.o » et « comgps.h » sont à copier dans le répertoire qui contient le fichier de décodage de la trame

Ensuite, il faut inclure le fichier « comgps.h » dans le code source de votre programme en ajoutant la ligne suivante :

```
#include "comgps.h"
```

Compiler votre programme en utilisant la commande suivante :

```
gcc -o gps votreprogramme.c comgps.o
```

Le fichier exécutable résultant « gps » est compilé et lié avec la bibliothèque. On parle de librairie statique et non dynamique.