teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun Polytechniques

Année : 2015

Filière : TSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

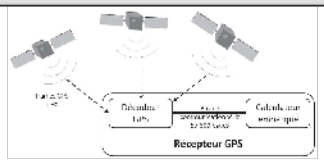
L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

	Informatique pour tous	
	Info CCP - TSI 2015 Proposition de correction UPSTI	Version 0.12
Proposition de corrigé UPSTI		

Remarques générales sujet :

Il s'agit d'un sujet d'informatique contextualisé autour du traitement des informations issues d'un décodeur GPS. Le sujet ne présente pas de difficulté majeure et à part les bases de données, l'ensemble des points du programme de l'Informatique pour Tous est abordé.

Ce corrigé est proposé en Python.

1. Acquisition des trames NMEA via une liaison série

1.1 Liaison série

Q1. Pour obtenir le nombre d'octets et en ne considérant que la latitude, la longitude avec les deux précisions et la vitesse en nœud, on obtient :

$$\begin{aligned} nb\_octets = & \\ & 9(latitude) + \\ & 1(précision) + \\ & 10(longitude) + \\ & 1(précision) + \\ & 4(vitesse\,en\,noeuds) \\ & = 25\,octets \end{aligned}$$

Q2.

1. Chaque caractère est composé d'un octet précédé d'un bit de start et suivi d'un bit de stop. On arrive ainsi à un total de 10 bits par caractère. L'ensemble représente 76 caractères soit 760 bits. La vitesse de transmission est de 57600 bits / s. On a alors :

$$durée\,de\,transmission = \frac{760}{57600} = 0,0132\,s$$

2. Le récepteur reçoit une trame toutes les secondes des satellites. La durée de transmission est de 0,0132 s. Le débit est suffisant pour traiter en temps réel les trames provenant des satellites.

1.2 Lecture des trames GPS

Q3. L'instruction permettant d'initialiser le port série numéroté 0 avec une vitesse de 57600 bauds et renseignant la variable identifiant est :

$$identifiant = initSerie(0,57600)$$

Q4.

1. Le dernier caractère de la trame est le caractère <LF>. Son code décimal peut se déterminer à partir de l'annexe 2 et sa valeur est 10.

2. La fonction à utiliser pour obtenir un caractère à partir de son code ASCII est la fonction chr().

Q5. Il suffit de coder l'algorithme proposé.

Algorithme : lecture et affichage d'une trame

Fonction lireTrame(*Paramètre à définir* **)**
- **Entrée :** *à définir*
- **Sortie :** *à définir*
- trame ← chaîne de caractères vide
- lu ← 0
- **Tant que** lu ≠ nombre entier correspondant au caractère de fin de trame en ASCII **faire**
  - lu ← lire octet série correspondant à un caractère
  - ajouter en fin de trame le caractère correspondant à lu
- **Fin Tant que**
- **Afficher** (trame)
- **Retourner** (trame)

Fin Fonction lireTrame

```
def lireTrame (identifiant):
    """
    Fonction de lecture de trame qui permet de retourner une trame à partir
    d'un port série préalablement ouvert
    Entrée :
        * identifiant : int - identifiant d'un port préalablement ouvert
    Sortie :
        * Trame : str - Trame lue sur le port série
    """
    trame=""
    lu=0
    while (lu!=10):
        lu=lireSerie(identifiant)
        trame = trame + chr(lu)
    print (trame)
    return (trame)
```

1.3 Enregistrement dans un fichier

Q6. On peut proposer le programme suivant dont les actions consécutives sont d'ouvrir un fichier et d'enregistrer en continu dans ce fichier les trames reçues. Il est à noter que compte tenu du sujet et de l'orientation proposée dans la résolution de cette question, la fermeture (propre) du fichier est impossible.

```
f=open('tramesNMEA.txt','w')
identifiant=initSerie(0,57600)
while (True):
    trame_courante = lireTrame(identifiant)
    f.write(trame_courante)
f.close()
```

La fonction open() retourne un objet qui servira à appeler les méthodes write() et close(). Dans la solution proposée, la méthode close() ne sera jamais utilisée car le programme ne présente pas de fin. La manière de le stopper est de le « breaker » par CTRL + C.

Note : On peut regrouper les deux lignes dans la boucle en :

```
f=open('tramesNMEA.txt','w')
identifiant=initSerie(0,57600)
while (True):
    f.write(lireTrame(identifiant))
f.close()
```

2. Exploitation du fichier de trames

2.1 Extraction d'une trame

Q7.

1. A la lecture du contenu de la fonction, on se rend compte qu'il n'y a pas de variant de boucle. Le contenu de la variable i n'est pas incrémenté. Il reste à 0. Le résultat du test proposé pour sortir de la boucle ne peut pas devenir vrai.
2. La ligne de code à insérer est une incrémentation de la variable i. Elle se situe entre les lignes 16 et 17 actuelles.

```
def isCharInString(car, chaine):
    """
    Recherche si un caractère est dans une
    chaîne de caractères.
    Entrées :
        * car : str, caractère
        * chaine : str, chaîne de caractères
    Sortie :
        * un booléen : True si les caractère est
        présent, False sinon
    """
    n = len (chaine)
    i=0
    while i<n:
        if chaine[i]==car:
            return True
    return False
```

```
def isCharInString(car, chaine):
    """
    Recherche si un caractère est dans une
    chaîne de caractères.
    Entrées :
        * car : str, caractère
        * chaine : str, chaîne de caractères
    Sortie :
        * un booléen : True si les caractère est
        présent, False sinon
    """
    n = len (chaine)
    i=0
    while i<n:
        if chaine[i]==car:
            return True
        i=i+1
    return False
```

Q8. Cette fonction est assez voisine de la fonction précédente. Il existe plusieurs solutions pour le retour de la fonction si le caractère recherché n'est pas trouvé dans la chaîne de caractères.

- On peut retourner -1. Cette solution est assez voisine d'une solution classique en C, langage qui n'admet pas des types retour différents.
- On peut retourner False. Cette solution repose sur la possibilité de typage dynamique mise en place en Python.

```python
def cherchePremiereOccurence(car, chaine):
    """
    Retourne l'indice de la première occurence d'un caractère.
    Entrées :
        * car : str, caractère
        * chaine : str, chaîne de caractères
    Sortie :
        * i : int (si l'indice existe)
        * -1 : int (si l'indice n'existe pas :  solution 1)
        * false : booléen (si l'indice n'existe pas : solution 2)
    """
    n=len(chaine)
    i=0
    while (i<n):
        if chaine[i]==car:
            return i
        i=i+1
    #return -1      # solution 1
    return False    # solution 2

### Tests Fonction cherchePremiereOccurence() ###
print (cherchePremiereOccurence ('a', 'ma chaine'))
print (cherchePremiereOccurence ('b', 'ma chaine'))
```

Q9.

1. Il faut que les caractères '$' et '*' soient présents dans la chaîne de caractères. L'indice du caractère '$' doit être inférieur à l'indice du caractère '*'. On peut également chercher si le caractère '$' est bien en position 0 (en python) indiquant qu'il s'agit d'une chaîne commençant par le caractère autorisé de début de chaîne.
2. La réponse proposée est algorithmique. (Elle tient cependant compte que les chaînes de caractères sont indexées à partir de l'indice 0 en Python.)

Fonction extraireTrame (NMEA : type str)

Début

* Entrée : NMEA str : Trame encadrée par les caractères « $ » et « * »
* Sortie : NMEA_extraite : str
  Chaine vide si les conditions d'extractions ne sont pas remplies.
  Contenu de la chaine encadrée par les caractères « $ » et « * » si les conditions d'extraction sont remplies.

NMEA_extraite ← ""
Si debut ← cherchePremiereOccurence (NMEA,' $') = 0 ET
fin ← cherchePremiereOccurence (NMEA,' *') >0 **Alors**
Pour i de 1 à fin **exclus** par pas de 1
NMEA_extraite ← NMEA_extraite + NMEA[i]
FinPour
FinSi

Retourner (NMEA_extraite)

Fin

L'extraction commence à partir de la valeur d'indice 1 et se termine à la valeur de l'indice fin sans prendre en compte cette information car à cette position, la donnée stockée si elle existe est le caractère « * ».

A titre informatif, on peut donner une implémentation possible de cet algorithme. Les affectations des variables *debut* et *fin* sont extraites de la structure de sélection pour être réalisées avant le test.

```
NMEA='$ma chaine A* pas extrait'

def extraireTrame (NMEA):
    NMEA_extraite = ""
    debut = cherchePremiereOccurence ('$',NMEA)
    fin = cherchePremiereOccurence ('*',NMEA)
    if (debut == 0) and (fin > 0):
       for i in range (1,fin) :
           NMEA_extraite = NMEA_extraite + NMEA[i]
    return (NMEA_extraite)

NMEA_extraite=extraireTrame (NMEA)
print NMEA_extraite
```

Note : A partir des informations contenues dans l'annexe 6 du sujet, on peut également proposer une autre implémentation de la fonction qui se passe de la boucle interne.

```
NMEA='$ma chaine A* pas extrait'

def extraireTrame (NMEA):
    NMEA_extraite = ""
    debut = cherchePremiereOccurence ('$',NMEA)
    fin = cherchePremiereOccurence ('*',NMEA)
    if (debut == 0) and (fin > 0):
        NMEA_extraite = NMEA[1:fin]
    return (NMEA_extraite)

NMEA_extraite=extraireTrame (NMEA)
print NMEA_extraite
```

2.2 Structure de données

Q10.

1. Pour compléter le tableau, il suffit de compter les différentes informations présentes dans la trame en prenant en compte le langage cible.

Pour l'indice de fin, il est possible de proposer deux réponses différentes à cette question suivant que ce dernier est comptabilisé inclus ou exclus.

	Latitude	Orientation	Longitude	Orientation	Vitesse
Index Début	19	29	31	42	44
Index fin (inclus)	27	29	40	42	47
Index fin (exclus)	28	30	41	43	48

La deuxième version (index de fin exlus) permet d'utiliser plus simplement la méthode d'extraction d'une sous chaîne de caractères dans une chaine de caractères proposée dans l'annexe 6.

2. La fonction creationPointGPS() réalise l'extraction des informations utiles dans la trame NMEA_extraite.

On peut proposer « une version longue » qui détaille les différentes opérations d'extraction.

```
def creationPointGPS(NMEA_extraite):
    chaine_latitude = NMEA_extraite[19:28]
    chaine_orientation_lat = NMEA_extraite[29:30]

    latitude = conversionLongLat2Min (chaine_latitude,chaine_orientation_lat)

    chaine_longitude = NMEA_extraite[31:41]
    chaine_orientation_lg = NMEA_extraite[42:43]

    longitude = conversionLongLat2Min (chaine_longitude,chaine_orientation_lg)

    vitesse = NMEA_extraite[44:48]

    return [latitude, longitude, vitesse]
```

Dans cette version, la latitude et la longitude sont transformées en valeurs numériques de type flottant alors que ce n'est pas le cas de la vitesse.
Si on veut transformer également la vitesse en valeur numérique de type flottant, il faut utiliser une opération de cast.
La deuxième version proposée est la version courte adaptée au Document Réponse tenant compte de la remarque précédente.

```
def creationPointGPS(NMEA_extraite):
    latitude= conversionLongLat2Min (NMEA_extraite[19:28],NMEA_extraite[29:30])
    longitude= conversionLongLat2Min (NMEA_extraite[31:41],NMEA_extraite[42:43])
    vitesse = float(NMEA_extraite[44:48])

    return [latitude, longitude, vitesse]
```

2.3 Affichage du fichier

Q11. L'affichage est désormais possible à partir du fichier de trames de nom tramesNMEA.txt. Il faut réaliser les opérations suivantes :

Ouvrir le fichier.
Tant qu'il reste des lignes
    Lire une ligne.
    Extraire la sous trame utile de la trame.
    Obtenir le point GPS et l'afficher sous la forme demandée.
FinTantQue
Fermer le fichier.

```
f=open('tramesNMEA.txt','r')
fin=0
while (fin!=1):
    NMEA=f.readline()
    if NMEA=="":
        fin=1
    else :
        trame_extraite=extraireTrame(NMEA)
        pointGPS=creationPointGPS(trame_extraite)
        print (str(pointGPS[0])+'\t'+str(pointGPS[1])+'\t'+str(pointGPS[2]))
f.close()
```

Note : L'objet f, renseigné par la fonction open() contient un itérateur qui permet de parcourir le fichier ligne par ligne. L'utilisation de ce dernier permet de simplifier le code précédent en évitant le recours à l'utilisation de la méthode readline().

```
f=open('tramesNMEA.txt','r')
# L'utilisation de la méthode readline() sur l'objet f n'est plus utile.
for ligne in f :
    trame_extraite=extraireTrame(ligne)
    pointGPS=creationPointGPS(trame_extraite)
    print (str(pointGPS[0])+'\t'+str(pointGPS[1])+'\t'+str(pointGPS[2]))
f.close()
```

3. Structure avancée de données

Q12. La comparaison doit juste se faire sur la vitesse qui se trouve en position 2 dans un pointGPS. Tout le reste est inchangé.

```
def percolateUp(tas):
    index = len(tas)-1
    val = tas[index]
    while (index>0):
        index_pere = int ((index-1)/2)
        pere = tas[index_pere]
        if pere > val :
            break
        else :
            tas[index] = pere
            tas[index_pere] = val
            index = index_pere
```

```
def percolateUp(tas):
    index = len(tas)-1
    val = tas[index]
    while (index>0):
        index_pere = int ((index-1)/2)
        pere = tas[index_pere]
        if pere[2] > val[2] :
            break
        else :
            tas[index] = pere
            tas[index_pere] = val
            index = index_pere
```

Q13. On souhaite insérer un point GPS dans le tas (qui est réalisé à partir d'une liste). Pour cela, dans la fonction insererPointGPS(), il suffit juste d'ajouter le point GPS dans le tas puis de faire appel à la fonction percolateUp () définie précédemment qui va le placer 'au bon endroit'.

```
def insererPointGPS(tas, pointGPS):
    tas.append(pointGPS)
    percolateUp(tas)
```

Note sur le Document Réponse proposé : Dans le Document Réponse à cette question, il manque de la place pour indiquer le nom des paramètres passés à la fonction entre les deux parenthèses.

Q14. Question 14

1. Dans le tas binaire, la donnée se trouvant en haut de l'arbre contient la valeur la plus grande suivant le critère de tri qui est mis en place dans la fonction percolateUp(). L'extraction du point possédant la vitesse maximale se fait juste à partir d'une extraction de la donnée en position 0.

```
def extractMax(tas):
    return (tas[0])
```

Note sur le Document Réponse proposé : La même remarque que précédemment sur la taille de la zone des parenthèses dans le Document Réponse peut être faite.

2. L'avantage de cette structure de donnée réside justement dans l'extraction de la valeur maximale qui se fait en temps constant. On a alors une complexité indépendante de N, soit $O(1)$.

4. Interface graphique

4.1 Etude du principe de calcul des vitesses lissées et association d'un code couleur à chaque vitesse lissée.

Q15.

1. Compte tenu de la méthode mise en place. L'index absolu de départ est 3 car il faut disposer des deux valeurs précédentes indexées 1 et 2. L'index absolu de fin est 9, car il faut disposer de la valeur indexée 10.
2. Il sera donc possible de calculer 7 moyennes glissantes différentes.
3. On peut étendre le résultat précédent à une liste de N vitesses. Dans ce cas le nombre de moyennes glissantes susceptibles d'être déterminées est de N-3.

Q16.

Note : La réponse à cette question est d'ordre algorithmique et la détermination de la valeur des index se fait sur les valeurs d'index absolu définies dans le tableau Table 8.

1. La valeur de index_i_absolu_debut est la même que celle de l'index absolu de départ à savoir 3. La valeur de index_i_absolu_fin est la même que celle de l'index absolu de fin à savoir 9.
2. L'index_j_absolu doit commencer deux positions avant celle de l'index i et se terminer une position plus loin. Soit index_j_absolu_debut = i – 2 et index_j_absolu_fin = i+1.
3. La réponse est d'ordre algorithmique :

```
Fonction moyenneGlissante(vit)
    Entrée : vit : liste de vitesses instantanées
    Sortie : res : liste des vitesses lissées sur 4 valeurs
    res ← liste vide
    Pour i allant de index_i_absolu_début à index_i_absolu_fin inclus par pas de 1
        Som ← 0
        Moy ← 0
        Pour j allant de index_j_absolu_début à index_j_absolu_fin inclus par pas de 1
            .....
            .....
        Fin Pour
        Ajouter Moy dans la liste res
    Fin Pour
    Retouner res
Fin Fonction moyenneGlissante
```

```
Fonction moyenneGlissante (vit)
Début
    * Entrée : vit :liste de vitesses instantanées
    * Sortie : res : liste des vitesses lissées sur 4 valeurs

    res ← liste vide"
    Pour i allant de index_i_absolu_début à index_i_absolu fin inclus par pas de 1
        Som ← 0
        Moy ← 0
        Pour j allant de index_j_absolu_début à index_j_absolu fin inclus par pas de 1
            Som ← Som + vit[j]
            Moy ← Moy / 4
        FinPour
        Ajouter Moy dans la liste res
    FinPour
    Retourner res
Fin
```

Note sur le Document Réponse proposé : Dans la solution proposée sur le Document Réponse et avec un calcul simple de la moyenne (en divisant par 4), le calcul de la moyenne courante est faux sauf quand les quatre valeurs ont été prises en compte dans la boucle, c'est à dire en sortie de boucle. On peut proposer une autre solution.

```
Fonction moyenneGlissante (vit)
Début
    * Entrée : vit :liste de vitesses instantanées
    * Sortie : res : liste des vitesses lissées sur 4 valeurs

    res ← liste vide"
    Pour i allant de index_i_absolu_début à index_i_absolu fin inclus par pas de 1
        Som ← 0
        Moy ← 0
        Pour j allant de index_j_absolu_début à index_j_absolu fin inclus par pas de 1
            Som ← Som + vit[j]
        FinPour
        Moy ← Moy / 4
        Ajouter Moy dans la liste res
    FinPour
    Retourner res
Fin
```

Q17. Pour associer les codes couleurs 255 et 0 aux vitesses maximale et minimale, il suffit juste dans un premier temps de procéder à une recherche des deux extrema ; à savoir la vitesse maximale et la vitesse minimale pour associer à ces vitesses les codes couleurs proposés.

Q18. L'association est linéaire et il suffit de résoudre un système.

$$C_{Couleur}(v) = a.v + b \text{ soit}$$

$$\begin{cases} 255 = a.v_{max} + b \\ 0 = a.v_{min} + b \end{cases} \quad donc\ a = \frac{255}{v_{max} - v_{min}}\ et\ b = -a.v_{min} = -\frac{255}{v_{max} - v_{min}}.v_{min}$$

$$donc\ C_{Couleur}(v) = a.v + b = \frac{255}{v_{max} - v_{min}}.v - \frac{255}{v_{max} - v_{min}}.v_{min}$$

$$Finalement\ C_{Couleur}(v) = a.v + b = \frac{255}{v_{max} - v_{min}}.(v - v_{min})$$

4.2 Application : affichage du tracé

Q19. Dans cette question, il faut juste faire un test sur les valeurs par défaut de la fonction pour déterminer s'il faut appeler la fonction tracePoint() ou la fonction traceSegment().

```
def ajoutePoint (id_carte, lat_Pt_1, long_Pt_1, lat_Pt_2=0, long_Pt_2=0, coul=255):
    if ((lat_Pt_2==0) and (long_Pt_2==0) and (coul==255)):
        tracePoint(id_carte, lat_Pt_1, long_Pt_1)
    else:
        traceSegment(id_carte, lat_Pt_1, long_Pt_1, lat_Pt_2, long_Pt_2, coul)
```

Q20. Il s'agit d'une question de synthèse permettant de proposer l'écriture du programme principal destiné à afficher les points sur la carte.

```
# Saisie du nom de fichier
non_fichier = input("Saisir le nom du fichier ") ###

# Saisie du nom de la carte
carte = input("Saisir le nom de la carte : ")

# Saisie de l'échelle (stockée sous forme de chaîne de caractères)
echelle = input ("Saisir l'echelle : ")
echelle = float(echelle) # Conversion de la chaine de caractères en flottant

# Saisie de la latitude de l'origine de la carte
lat0 = float(input("Saisir la latitude de l'origine :"))
# Saisie de la longitude de l'origine de la carte
long0 = float(input("Saisir la longitude de l'origine :"))
# Récuperation des points à partir du fichier de trames tramesNMEA.txt
liste_pointGPSlisse = trameGPS2liste(non_fichier) ###

# Initialisation de la carte
initCarte(carte, echelle,float(lat0), float(long0)) ###

# Affichage de la carte
afficheCarte(carte) ###

# Ajout de points sur la carte à partir de la liste de points pts
for i in range (0, len (liste_pointGPSlisse)-1):
    ajoutePoint(carte,liste_pointGPSlisse[i][0],liste_pointGPSlisse[i][1], \
    liste_pointGPSlisse[i+1][0],liste_pointGPSlisse[i+1][1], \
    liste_pointGPSlisse[i][3])   ###
```

Note sur le Document Réponse proposé : Dans le Document Réponse, il est fait mention de l'opération de cast par la « fonction » flt (). Une solution possible pour réaliser cette opération de cast est l'utilisation de la « fonction » float() qui est disponible en Python.

Q21. Le sujet a permis d'appréhender les différentes étapes nécessaires au tracé d'un itinéraire sur une carte.

Pour cela, le traitement commence par acquérir les données reçues par le décodeur GPS pour les sauvegarder dans un fichier. A partir de ce dernier, le sujet a cherché à proposer une méthode d'extraction des données utiles des trames sauvegardées.

La deuxième partie du sujet met en évidence les traitements nécessaires pour déterminer la vitesse sur le parcours à partir d'un calcul de moyenne glissante afin de s'affranchir des erreurs de mesure inévitables dans ce genre d'acquisition.

Le sujet se termine par l'écriture du programme principal d'affichage des données sur une carte.

On peut noter que l'utilisation d'un tas binaire permet un accès en temps constant au point possédant la plus grande vitesse. Il faut cependant préciser que l'utilisation de ce tas binaire pour tracer le parcours aurait exigé que les données sauvées soient horodatées et qu'une recherche soit alors effectuée sur cette information.