GM3 Nathalie Chaignaud

DS2 - Algorithmes et Structures de Données

Vendredi 8 Janvier 2021 - Correction

1. Pile – 5 pts

`quick-select` effectue la recherche du $k^{ème}$ plus petit entier `e` dans le tableau d'entiers `t` entre les indices `inf` et `sup`.

```
Const max = 10
Type tab = tableau [1..max] d'entier

Procédure quickSelect(E/S t : tab ; E k, inf, sup : entier ; S e : entier)
Var p : entier
Début
  Si inf=sup
    Alors e←t[inf]
    Sinon
      partitionner(t,inf,sup,p)  (* où p est l'indice du pivot *)
      Si k=p
        Alors e←t[p]
        Sinon Si k<p Alors quickSelect(t,k,inf,p-1,e) {@1}
                     Sinon quickSelect(t,k,p+1,sup,e) {@2}
              FinSi
      FinSi
  FinSi
Fin
```

t : 4 10 8 18 12 2 16 0 14 6

Simuler la pile sur l'appel `quickSelect(t,7,1,max,e)` {@0} en donnant aussi les valeurs intermédiaires de t après l'appel de `partitionner` (donné dans le résumé).

@2, k = 7, inf = 5, sup = 9,	t = 2	0	4	6	12	8	16	10	14	18
@1, k = 7, inf = 4, sup = 9,	t = 2	0	4	6	12	8	16	10	14	18
@2, k = 7 , inf = 4, sup = 10,	t = 2	0	4	18	12	8	16	10	14	6
@0, k = 7, inf = 1, sup = 10,	t = 4	10	8	18	12	2	16	0	14	6

```
1. partitionner(t=4,10,8,18,12,2,16,0,14,6; d=1,f=10)→ (2,0,4,18,12,8,16,10,14,6);p=3
2. partitionner(t=2,0,4,18,12,8,16,10,14,6; d=4,f=10)→(2,0,4,6,12,8,16,10,14,18);p=10
3. partitionner(t=2,0,4,6,12,8,16,10,14,18 ; d=4,f=9)→ (2,0,4,6,12,8,16,10,14,18);p=4
4. partitionner(t=2,0,4,6,12,8,16,10,14,18 ; d=5,f=9)→ (2,0,4,6,10,8,12,16,14,18);p=7

e=12
```

2. Représentations d'un polynôme – 5 pts

Pour manipuler des polynômes, on peut utiliser plusieurs structures de données, dont les tableaux et les listes chaînées. Par exemple, on peut utiliser ces 2 représentations :

- ```
  Const degmax = 20
  Type poly1 = tableau [0..degmax] de réel
  ```
  où la valeur d'une case du tableau représente le coefficient (réel) du terme dont le degré (entier ≤ `degmax`) est donné par l'indice de la case.
- ```
  Type terme = Enregistrement
                    coef : réel
                    deg : entier
                    suiv : ^terme
               FinEnregistrement
       poly2 = ^terme
  ```
  où les termes du polynôme sont chaînés et triés dans l'ordre croissant des degrés.

On souhaite pouvoir passer d'une représentation à l'autre.

2.1. Ecrire en pseudo-langage une fonction `poly1To2` qui fait la conversion du type `poly1` vers le type `poly2`.

```
Procédure poly1To2(p1 : poly1) : poly2
Var p2, q :  poly2
Début
 p2 ← nil
 Pour i←0 à degmax inc +1 faire
 {la liste chaînée ne contient pas les termes « nuls »}
      Si p1[i]≠0
         Alors t ← allouer(terme)
               t^.deg ← i
               t^.coef ← p1[i]
               Si p2=nil Alors p2 ← t  {Cas de la tête de liste}
                                 q ← t
                         Sinon q^.suiv ← t
                                 q ← q^.suiv
               FinSi
      FinSi
  FinPour
  {On n'oublie pas de mettre nil à la fin de p2, q pointe sur le dernier terme}
  Si p2 ≠ nil
      Alors q^.suiv ← nil
  FinSi
  retourner(p2)
Fi
```

2.2. Ecrire en pseudo-langage une fonction `poly2To1` qui fait la conversion du type `poly2` vers le type `poly1`.

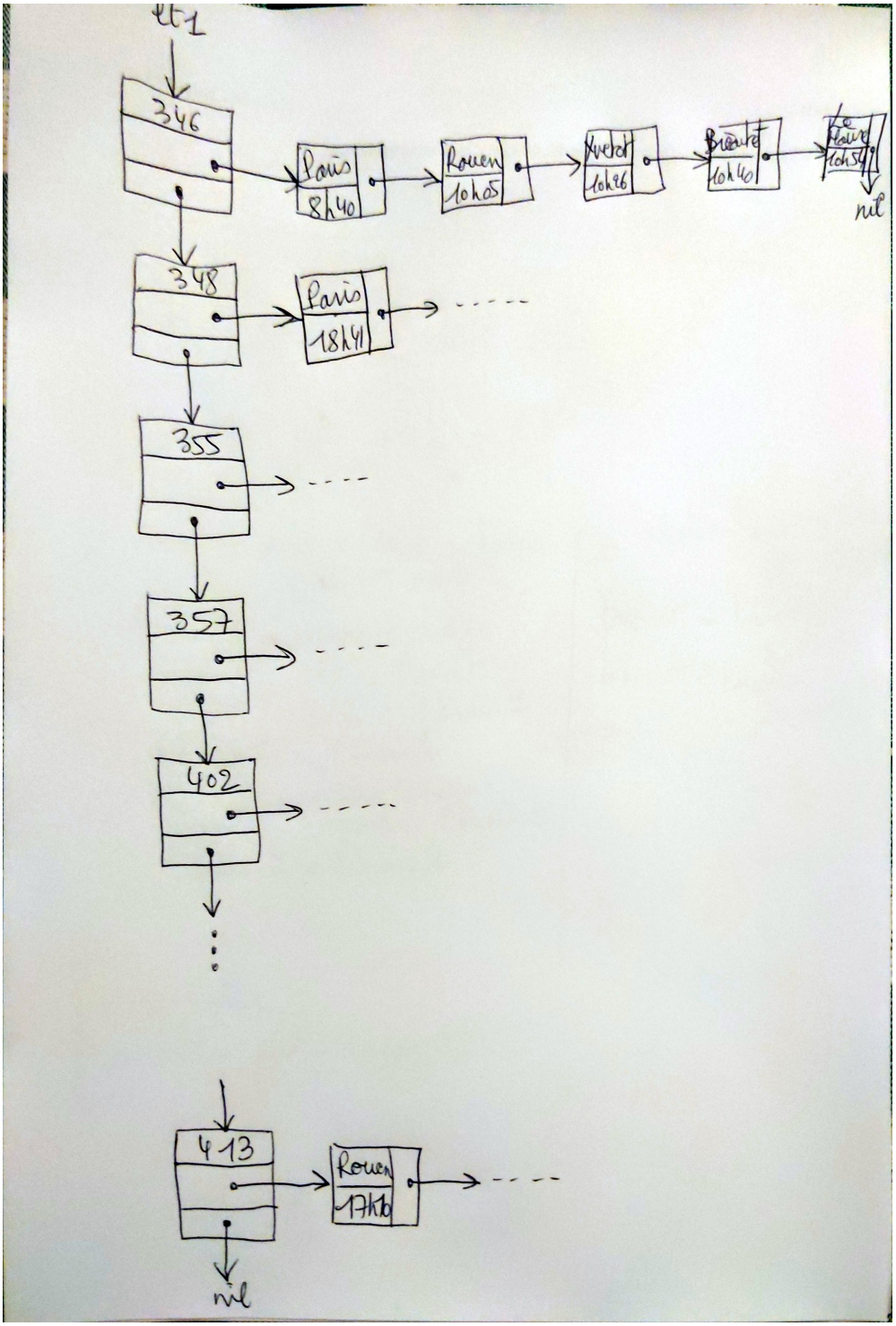
```
Procédure poly2To1(p2 : poly2) : poly1
Var p1 :  poly1
Début
 {On initialise le tableau à 0}
  Pour i←0 à degmax inc +1 faire
      p1[i]←0
  FinPour
 {On met à jour uniquement les cases correspondant aux termes dela liste chaînée}
  TantQue p2≠nil faire
      p1[p2^.deg] ← p2^.coef
      p2 ← p2^.suiv
  FinTantQue
retourner(p1)
Fin
```

3. Gestion de trains – 10 pts

On souhaite pouvoir gérer des trains avec leurs différents arrêts aux stations et leurs horaires. On propose la structure de données suivante pour représenter la liste de trains :

```
Type  arret = Enregistrement
                  nom, horaire : chaîne
                  suiv : ^arret
              FinEnregistrement
      train1 = Enregistrement
                  num : chaine
                  la : ^arret
                  suiv : ^train1
             FinEnregistrement
      ltrain1 = ^train1
```

3.1. Faire un dessin de la structure de données avec les données suivantes.



3.2. Ecrire en pseudo-langage la procédure `supprimeTrain` qui supprime de la liste `lt1` un train existant donné par son numéro, ainsi que tous les arrêts. **Faire un dessin.**

```
Procédure supprimeTrain(E num : chaîne ; E/S lt1 : ltrain1)
Var l, t : ltrain1
    p, q : ^arret
Début
  l ← lt1
  Si l^.num=num {Cas où le train à supprimer est le premier}
      Alors t ← l
      Sinon
      {On positionne t sur le train qu'on souhaite supprimer et l sur le précédent}
            TantQue l^.suiv^.num≠num Faire
               l←l^.suiv
            FinTantQue
            t ← l^.suiv
  FinSi
  {On supprime tous les arrêts du train pointé par t}
  p ← t^.la
  TantQue p≠nil Faire
      q ← p^.suiv
      récupérer(p)
      p ← q
  FinTantQue
  {On supprime maintenant le train pointé par t}
  Si l=lt1
      Alors lt1 ← l^.suiv {Cas où le train à supprimer est le premier}
            récupérer(l)
      Sinon l^.suiv ← t^.suiv
            récupérer(t)
  FinSi
Fin
```

3.3. Ecrire en pseudo-langage la fonction `donneTrainStation` qui renvoie un pointeur sur une liste de trains (non vide) passant par une station (existante et donnée par son nom). **Faire un dessin.**

```
Type  train2 = Enregistrement
                  num : chaîne
                  suiv : ^train2
               FinEnregistrement
      ltrain2 = ^train2

Fonction donneTrainStation(lt1 : ltrain1, nom : chaîne) : ltrain2
Var l2, q : ltrain2
    la : ^arret
Début
  l2 ← nil {l2 pointe sur la liste des trains passant par la station demandée}
  {On parcourt toute la liste des trains lt1}
  TantQue lt1≠nil Faire
    la ← lt1^.la
    {Pour chaque train, on regarde si la station fait partie des arrêts du train}
    TantQue la≠nil et la^.nom≠nom Faire
      la ← la^.suiv
    FinTantQue
    Si la^.nom=nom {Si on  a trouvé la station, alors on ajoute le train à l2}
      Alors t ← allouer(train2)
            t^.num ← lt1^.num
            Si l2=nil  {Cas de la tête de liste}
              Alors l2 ← t
                    q ← t
              Sinon q^.suiv ← t
                    q ← q^.suiv
            FinSi
    Finsi
    lt1 ← lt1^.suiv
  FinTantQue
  q^.suiv ← nil
  retourner(l2)
Fin
```

3.4. Ecrire en pseudo-langage la fonction `donneTrains` qui renvoie un pointeur sur une liste de trains passant successivement par la station1 puis la station2 (données respectivement par `nom1` et `nom2`). La liste peut être égale à `nil`. **Faire un dessin.**

```
Fonction donneTrains(lt1 : ltrain1, nom1, nom2 : chaîne) : ltrain2
Var l2, q : ltrain2
    la, ps : ^arret
Debut
  l2 ← nil {l2 pointe sur la liste des trains passant par les stations demandées}
  {On parcourt toute la liste des trains lt1}
  TantQue lt1≠nil Faire
    la ← lt1^.la
    {Pour chaque train, on regarde si nom1 puis nom2 font partie des arrêts du train}
    TantQue la≠nil et la^.nom≠nom1 Faire
      la ← la^.suiv
    FinTantQue
    Si la^.nom=nom1 {Si on  a trouvé nom1, alors on cherche nom2}
      Alors ps ← la
            la ← la^.suiv
            TantQue la≠nil et la^.nom≠nom2 Faire
               la ← la^.suiv
            FinTantQue
            Si la^.nom=nom2 {Si on  a trouvé nom2, alors on ajoute le train à l2}
               Alors t ← allouer(train2)
                     t^.num ← lt1^.num
                     Si l2=nil  {Cas de la tête de liste}
                        Alors l2 ← t
                              q ← t
                        Sinon q^.suiv ← t
                              q ← q^.suiv
                     FinSi
            Finsi
    FinSi
    lt1 ← lt1^.suiv
  FinTantQue
  q^.suiv ← nil
  Retourner(l2)
Fin
```

3.5. Ecrire en pseudo-langage une procédure `créeTabStation` qui, à partir d'un tableau de `n` chaines représentant les noms de station (`tch`), crée un tableau de stations (`tst`) où chacune d'elles pointe sur la liste des trains qui y passent. On utilisera la fonction `donneTrainStation` de la question 3.2. **Faire un dessin.**

```
Const max = 100
Type  station = Enregistrement
                  nom : chaîne
                  lt : ltrain2
                FinEnregistrement
      tab-st = tableau[1..max] de station
      tab-ch = tableau[1..max] de chaine

Procédure créeTabStation(E lt1 : ltrain1, tch : tab-ch, n : entier ; S tst : tab-st)
Var i : entier
Début
  Pour i← 1 à n inc +1 Faire
      tst[i].nom ← tch[i]
      tst[i].lt ←  donneTrainStation(lt1,tch[i])
  FinPour
Fin
```

3.6. Ecrire en pseudo-langage la fonction `donneStationMaxTrain` qui renvoie le nom de la station (la première trouvée) par laquelle passent le plus de trains. V**otre fonction doit être optimisée.**

```
Fonction donneStationMaxTrain(tst : tab-st, n : entier) : chaîne
Var i, nbt, maxt : entier
    pt : ltrain2
    noms : chaine
Début
  maxt ← 0
  {tst n'est pas vide et il existe au moins un train par station}
  Pour i←1 à n inc +1 Faire
      pt ← tst[i].lt
      nbt ← 0
      TantQue pt≠nil Faire
         nbt ← nbt+1
         pt ← pt^.suiv
      FinTantQue
      Si nbt>maxt
         Alors maxt ← nbt
               noms ← tst[i].nom
      FinSi
  FinPour
Retourner(noms)
Fin
```