Algorithmes et Structures de Données

Mardi 17 Novembre 2020

Durée 1H30 – Cours et TD NON autorisés

1. Réussite - 12 pts

Soit un tableau de 11 cases rempli aléatoirement des entiers de 1 à 9 et de deux valeurs nulles (égales à 0).
Le principe de la réussite est le suivant :

- Echanger une valeur nulle (symbolisant une place vide) avec le successeur de l'entier contenu dans la case précédant la valeur nulle. Si la première valeur du tableau est nulle, on l'échange alors avec la valeur 1.
- Recommencer ce processus jusqu'à ce
  - qu'on soit bloqué (les valeurs nulles sont toutes deux placées après le 9)
  - ou que les 9 valeurs soient placées dans les 9 premières cases du tableau.

Exemple

t :	2	4	0	6	8	7	3	0	9	1	5
t :	2	*4*	**5**	6	8	7	3	0	9	1	**0**
t :	2	**0**	5	6	8	7	*3*	**4**	9	1	**0**

1.1. Ecrire en pseudo-langage une fonction `Position` qui renvoie l'indice de la première bonne case vide du tableau , 0 sinon.
Une case vide est « bonne » si elle n'est pas précédée d'un 9 ou d'un 0 précédé d'un 9.
1.2. Ecrire en pseudo-langage une procédure `Déplace` qui utilise `Position` pour échanger la case vide avec l'entier successeur de celui contenu dans la case précédant cette case vide.
1.3. Ecrire en pseudo-langage une fonction `Vérifie` qui renvoie le caractère 'e' si on a abouti à un échec, 'r' si on a réussi, 'c' si on doit continuer à jouer.
1.4. Ecrire en pseudo-langage le programme complet qui donne les tableaux intermédiaires jusqu'à l'échec ou la réussite.
1.5. Ecrire en pseudo-langage une procédure `Sauvegarde` qui sauvegarde un tableau dans un fichier texte (une seule ligne). On dispose de la fonction `int2car` qui fait la traduction d'un chiffre en un caractère.
1.6. Ecrire en pseudo-langage une procédure `Charge` qui charge un tableau sauvegardé dans un fichier texte (une seule ligne). On dispose de la fonction `car2int` qui fait la traduction d'un caractère en un chiffre.

1.1. 2 pts

```
Fonction Position (n : entier, t : tab) : entier
Var i, res : entier
    trouvé : booléen
Début
res←0
Si t[1]=0
    Alors res←1
    Sinon Si t[2]=0
              Alors Si t[1]≠9 Alors res←2
                    FinSi
              Sinon i←3
                  trouvé←faux
                  TantQue (¬trouvé) et (i≤n) Faire
                      Si t[i]=0 et t[i-1]=0
                        Alors trouvé←t[i-2]≠9
                        Sinon Si t[i]=0 Alors trouvé←t[i-1]≠9
                              FinSi
                      FinSi
                      i←i+1
                  FinTantQue
                  Si trouvé Alors res←i-1
                  FinSi
         FinSi
FinSi
retourner(res)
Fin
```

1.2. 2 pts

```
Procédure Déplace (E n : entier , E/S t : tab)
Var p,j,nb : entier
Début
p←Position(n,t)
Si p≠0
   Alors Si p=1
            Alors nb←1
            Sinon nb←t[p-1]+1
         FinSi
         j←1
         TantQue t[j]≠nb Faire
            j←j+1
         FinTantQue
         t[p]←nb
         t[j]←0
Finsi
Fin
```

1.3. 2 pts

```
Fonction Vérifie (n : entier, t : tab) : car
Var i : entier
    res : char
Début
Si Position(n,t)=0
      Alors i←1
            TantQue (t[i]=i) et (i<=n-2) Faire
                  i←i+1
            FinTantQue
            Si (i=10)
                  Alors res←'r'
                  Sinon res←'e'
            FinSi
      Sinon res←'c'
FinSi
retourner(res)
Fin
```

1.4. 2 pts

```
Programme Réussite
Const n = 11
Type tab = tableau [1..n] d'entier
Var  t : tab
     v : car
     i : entier

      (* Déclaration des procédures et fonctions *)

Début
(* initilaisation et écriture du tableau initial *)
Pour i←1 à n inc +1 Faire
      lire(t[i])
      écrire(t[i],' ')
FinPour
v←vérifie(n,t)
TantQue v='c' Faire
      Déplace(n,t)
      Pour i←1 à n inc +1 Faire
            écrire(t[i],' ')
      FinPour
      v←vérifie(n,t)
FinTantQue
Si v='r'
      Alors écrire('Gagné !')
      Sinon écrire('Perdu !')
FinSi
Fin
```

1.5. 2 pts

```
Procédure Sauvegarde (E nomf : chaine , E t : tab)
Var f : FT
    i : entier
    c : chaine
Début
f ← CréerFichier(nomf)
c ← ''
Pour i←1 à n inc +1 Faire
      c ← concatèner(c,intTocar(t[i]))
      Si i≠n Alors c ← concatèner(c,' ')
      FinSi
FinPour
écrireChaine(f,c)
Fermer(f)
Fin
```

1.6. 2 pts

```
Procédure Charge (E nomf : chaine , S t : tab)
Var f : FT
    i,j : entier
    c : chaine
Debut
f ← OuvreEnLecture(nomf)
c ← lireChaine(f)
j ← 1
Pour i←1 à n inc +1 Faire
      t[i]← carToInt(c[j])
      j ← j+2
FinPour
Fermer(f)
Fin
```

2. Pile - 4 pts

```
Fonction f(x,y : entier) : entier
Var res : entier
Début
  Si (x=1) ou (y=x)
    Alors res ← 1
    Sinon res ← f(x,y-1) {@1} + f(x-1,y-1) {@2}
  FinSi
retourner(res)
Fin
```

Simuler la pile pour l'appel `écrire(f(3,5)){@0}` dans le programme principal.

~~@2, x=1, y=3 res=1~~
~~@2, x=1, y=2 res=1~~
~~@1, x=2, y=2 res=1~~
~~@1, x=2, y=3 res=1+1=2~~
~~@2, x=2, y=4 res=2+1=3~~
~~@2, x=1, y=2 res=1~~
~~@1, x=2, y=2 res=1~~
~~@2, x=2, y=3 res=1+1=2~~
~~@1, x=3, y=3 res=1~~
~~@1, x=3, y=4 res=1+2=3~~
~~@0, x=3, y=5 res=3+3=6~~

11 appels et retourne 6.

3. Récursivité - 4 pts

3.1. Ecrire en pseudo-langage une **fonction itérative** `Supprime-it` qui supprime toutes les occurrences d'une lettre dans un mot (chaine de caractères).
3.2. Ecrire en pseudo-langage une **fonction récursive** `Supprime-rec` qui supprime toutes les occurrences d'une lettre dans un mot (chaine de caractères). **Expliquer le principe.**

3.1. 1pt

```
Fonction supp-it(c : chaine, e : char) : chaine
Var res : chaine
      i : entier
Début
  res ← c
  i ← 1
  TantQue i<=lg(res) Faire
      Si res[i]=e
            Alors res ← supprimer(res,i,1)
            Sinon i ← i+1
      FinSi
  FintTantQue
retourner(res)
Fin
```

Variante optimale (sans supprimer) :

```
Fonction supp-it2(c : chaine, e : char) : chaine
Var res : chaine
    i, n : entier
Déb
  res ← ''
  n ← lg(c)
  Pour i← 1 à n inc +1 Faire
      Si c[i]<>e
            Alors res ← concaténer(res, c[i])
      FinSi
  FintPour
retourner(res)
Fin
```

3.2. 3pts

```
Fonction supp-rec(c : chaine, e : char ; i : entier) : chaine
Var res : chaine
Début
  si i>lg(c)
      Alors res ← ''
      Sinon Si c[i]=e
               Alors res ← supp-rec(supprimer(c,i,1),e,i)
               Sinon res ← concaténer(c[i],supp-rec(c,e,i+1))
            FinSi
  FinSi
retourner(res)
Fin
```

Variante sans i :

```
Fonction supp-rec2(c : chaine, e : char) : chaine
Var res : chaine
Début
  si c=''
      Alors res ← ''
      Sinon Si c[1]=e
               Alors res ← supp-rec2(supprimer(c,1,1),e))
               Sinon res ← concaténer(c[1],supp-rec2(supprimer(c,1,1),e))
            FinSi
  FinSi
retourner(res)
Fin
```