Université de M'hamad Bougara de Boumerdès
Faculté des Sciences — Département de Mathématiques
Deuxième Année Licence — Responsable du Module:
Recherche Opérationnelle — Mr. M. BEZOUI

Examen Final de Structure de Données

Barème: Exo01: 06Pt=2+2+2, Exo02: 06Pt=1.5+1.5+1.5+1.5, Exo03: 8Pt=1.5+1.5+1.5+1.5+2

Exercice $N°01$

1. Écrire une fonction "*miroir*" qui renvoie l'inverse des chiffres du nombre "N". Par exemple: le miroir de 5796 est 6975.

Algorithme 1 Inverser les chiffres d'un nombre

```
fonction miroir(N:entier):entier
variable res:entier;
début
si N<10 alors
   res ⟵ res * 10 + N;
sinon
   res ⟵ res * 10 + Nmod10;
   res ⟵ miroire(Ndiv10, res);
finsi
renvoyer(res);
fin;
```

2. Écrire une fonction récursive "*multiplier*" qui renvoie le produit d'un réel "x" fois un entier positif "p".

Algorithme 2 produit de x fois p

```
fonction multiplier(x:réèl,p:entier):réèl;
variable pro:entier;
début
si p=0 alors
   pro ⟵ x;
sinon
   pro ⟵ x + multiplier(x, p − 1);
finsi
renvoyer(pro);
fin;
```

3. Écrire une fonction récursive "*prod_scal*" qui renvoie le produit scalaire de deux vecteurs u et v de réels, donnés, de dimension N donnée.

Exercice $N°02$

Soit L,T, deux listes linéaires chaînées d'entiers, déclarées comme suit:

```
Type
   liste=enregistrement
         info: entier;
         suivant: ↑liste;
   fin enregistrement;
Variable
   L,T:↑liste;
```

Algorithme 3 produit scalaire de u et v

```
fonction prod_scal(u,v,N):réèl;
variable pro_s:réèl;
début
si N=0 alors
   pro_s ⟵ u[N] * v[N];
sinon
   pro_s ⟵ u[N] * v[N] + prod_scal(u, v, N − 1);
finsi
renvoyer(pro_s);
fin;
```

1. Écrire une fonction "$nombre\_elt(L :\uparrow liste)$" qui renvoie le nombre d'éléments de la liste L.

Algorithme 4 nombre d'éléments de la liste L

```
fonction nombre_elt(L :↑ liste):entier;
variable i:entier;
début
i⟵0; T⟵L;
tantque T<>NIL faire
   i⟵i+1;
   T ⟵ T ↑ .suivant;
fin tantque
renvoyer(i)
fin;
```

2. Écrire une fonction "$test(L :\uparrow liste)$" qui renvoie "Vrai" si la liste L est croissante, et "Faux" sinon.

Algorithme 5 Tester si la liste est croissante

```
fonction test(L :↑ liste):booléen;
variable b:booléen;
début
b⟵Vrai; T⟵L;
tantque T ↑ .suivant <> NIL et b faire
   si T ↑ .info > T ↑ .suivant ↑ .info alors
      b⟵Faux;
   finsi
   T ⟵ T ↑ .suivant;
fin tantque
renvoyer(b)
fin;
```

3. Écrire une fonction récursive "$Nb\_Oc(L :\uparrow liste, VAL)$" qui calcule le nombre d'occurrence d'une valeur "VAL" donnée dans la liste L.
4. Écrire une fonction "$Supp\_Doub(L :\uparrow liste)$" qui supprime les doublons de la liste L.

Exercice $N°03$

Soit P une pile d'entiers et F une file d'entiers:

1. Écrire une fonction "$nombre\_elt(P : pile)$" qui renvoie le taille de la pile P.
2. Écrire une fonction "$Somme(P : Pile)$" qui renvoie la somme des éléments de la pile P.
3. Écrire une fonction "$moyenne(P : Pile)$" qui renvoie la moyenne des éléments de la pile P.
4. Écrire une fonction "$Inverser(F : File)$" qui retourne l'inverse de la file F.

Algorithme 6 Calculer le nombre d'occurrence de la valeur Val

```
fonction Nb_Oc(L :↑ liste, VAL)
variable i:entier
début
si L = Nil alors
   Nb ← 0;
sinon
   si L↑.info = VAL alors
      Nb ← 1 + Nb_Oc(L↑.suivant, VAL);
   sinon
      Nb ← 0 + Nb_Oc(L↑.suivant, VAL);
   finsi
finsi
renvoyer(Nb)
fin;
```

Algorithme 7 Supprimer les doublons de la liste

```
fonction Supp_Doub(L :↑ liste)
variable T1:↑ liste;
début
T⟵L;
tantque T↑.suivant <> NIL faire
   P ← T; T1 ← T↑.suivant;
   tantque T1<>NIL faire
      si T↑.info = T1↑.info alors
         P↑.suivant ← T1↑.suivant;
      finsi
      P ← T1;
      T1 ⟵ T1↑.suivant;
   fin tantque
fin tantque
renvoyer(L)
fin;
```

5. Écrire une procédure "$Eclater(P : pile)$", permettant d'éclater une pile d'entiers en deux piles: P1 contenant des entiers négatifs et P2 contenant des entiers positifs.

Bon Courage
M. BeZoui

Algorithme 8 taille de la pile

```
fonction nombre_elt(P : pile) : entier
variable P1:Pile; i,val:entier;
début
créer_pile(P1); i ← 0;
tantque Non_Pile(P) faire
   dépiler(P,val);
   empiler(P1,val);
   i ← i + 1;
fin tantque
tantque Non_Pile(P1) faire
   dépiler(P1,val);
   empiler(P,val);
fin tantque
renvoyer(i)
fin;
```

Algorithme 9 somme des éléments de la pile P

```
fonction Somme(P : Pile) : entier
variable P1:Pile; s,val:entier;
début
créer_pile(P1); s ← 0;
tantque Non_Pile(P) faire
   dépiler(P,val);
   empiler(P1,val);
   s ← s + val;
fin tantque
tantque Non_Pile(P1) faire
   dépiler(P1,val);
   empiler(P,val);
fin tantque
renvoyer(s)
fin;
```

Algorithme 10 moyenne des éléments de la pile P

```
fonction moyenne(P : Pile) : rl
variable val:entier; m:réèl;
début
m ← Somme(P)/nombre_elt(P);
renvoyer(m)
fin;
```

Algorithme 11 inverse de la file F

```
fonction Inverser(F : File) : File
variable P1:Pile; val:entier;
début
créer_pile(P1);
tantque Non_File(F) faire
   défiler(F,val);
   empiler(P1,val);
fin tantque
tantque Non_Pile(P1) faire
   dépiler(P1,val);
   enfiler(F,val);
fin tantque
renvoyer(F)
fin;
```

Algorithme 12 éclater une pile d'entiers en deux piles

```
fonction Procdure(P : pile)
variable P1,P2:Pile; val:entier;
début
créer_pile(P1);
créer_pile(P2);
tantque Non_Pile(P) faire
  dépiler(P,val)
  si val<0 alors
    empiler(P1,val);
  sinon
    empiler(P2,val);
  finsi
fin tantque
fin;
```