Licence Sciences et Technologies	**ANNEE : 2004/2005** **SESSION DE MAI 2005** **ETAPE :** INF2, MAT2 et MAI4 **UE :** INF102 **Epreuve :** Initiation à l'algorithmique **Date :** **Heure :** 11 H **Durée :** 1 H 30 **Documents :** Tous documents interdits **Vous devez répondre directement sur le sujet qui comporte 4 pages.** **Insérez ensuite votre réponse dans une copie d'examen comportant tous les renseignements administratifs** **Epreuve de** Mme Delest.

Indiquez votre code **d'anonymat** : N° : .

La notation tiendra compte de la clarté de l'écriture des réponses.

Barème

- Question 1 – Connaissances générales : 4 points
- Question 2 – Compréhension d'algorithme de tris : 3 points
- Question 3 – Compréhension d'algorithme récursif : 3 points
- Question 4 – Manipulation de tableaux : 6 points
- Question 5 – Manipulation de listes : 4 points

Question 1. Cochez les affirmations qui sont correctes :

☒ La boucle *pour* comporte un test
☐ Les instructions d'une boucle *répéter* peuvent ne jamais être exécutées
☐ Une variable passée par *référence* à une fonction est recopiée dans l'environnement de la fonction.
☒ Une fonction *récursive* comporte au moins un test
☒ Il existe un algorithme de tri qui s'exécute toujours en *O(n)*
☒ Dans le tri à bulle, les *k* premiers éléments du tableau sont à leur place définitive après *k* itérations.
☐ Les méthodes « diviser pour régner » permettent toujours de diminuer la complexité.
☒ Dans une liste simplement chaînée de *n* éléments, la suppression du dernier élément se fait en *O(n)*

Question 2. Soit la fonction suivante :

```
fonction mystere(ref T:tableau[1..NMAX] d'entier,val N :entier) : entier ;
    var i,itmp :entier ;
    var tmp :tableau[1..N]d'entiers ;
    début
        init(tmp,0) ;
        i=1 ;
        itmp=0;
        tant que i<N faire
            si T[i] !=T[i+1] alors
                itmp=itmp+1 ;
                tmp[itmp]=T[i] ;
            finsi ;
            i=i+1 ;
        fintantque;
        si T[N] !=T[N-1] alors
            itmp=itmp+1 ;
            tmp[itmp]=T[N] ;
        finsi;
        copie(tmp,T,1,itmp ,1);
        retourner(itmp) ;
    fin ;
finfonction
```

où *init* et *copie* sont les fonctions décrites dans le cours.

1. On considère le tableau A de dimension 8 contenant la séquence 3,3,5,5,2,4,4,7. Quel est le résultat de l'appel mystere(A,8) ? Quel est l'effet de cet appel sur le tableau A ?
   Le résultat de l'appel est 5.
   Le tableau A est modifié car le tableau est passé par référence et on utilise la fonction copie qui copie le tableau tmp dans T. La tableau A contiendra 3,5,2,4,7.

2. Quelle sera la valeur entière maximale retournée par la fonction si le tableau lors de l'appel a pour taille N ? Illustrez par un exemple.
   La valeur maximale retournée sera N quand les valeurs contiguës contenues dans le tableau ne sont jamais égales. Par exemple pour N=5 et la séquence 3,5,2,1,4.

3. Que se passe-t-il si on remplace *ref* par *val* dans l'en-tête de la fonction ?
   Si on remplace ref par val alors le tableau sera copié dans le contexte d'exécution de la fonction. Par suite la fonction modifiera la copie et non le tableau fournit dans l'appel. Celui-ci ne sera donc pas modifié.

4. Donnez la complexité de cette fonction en nombre de tests ? $C^{<}(n)=C^{>}(n)=C(n)=O(n)$ ….

Question 3. Soit la fonction suivante :

```
fonction recurse(ref T:tableau[1..N] d'entier) :  entier ;
   fonction recurseRec(ref T:tableau[1..N] d'entier,val i :entier) : entier ;
      var j :entier ;
      début
         si T[i]==0 alors
               j=1 ;
         sinon
               j=0;
         finsi ;
         si i==N alors
            retourner(j) ;
         sinon
            retourner(j+recurseRec(T,i+1)) ;
         finsi
      fin
   finfonction ;
   début
      recurseRec(T,1) ;
   fin ;
finfonction
```

1. On considère le tableau A de dimension 4 contenant la séquence 3,0,1,0. Quel est le résultat de l'appel recurse(A) ? *Il n'y a pas retourner devant l'appel recurseRec(T,1) ; donc …rien …*
2. Donner la suite des appels récursifs provoqués par l'appel de la question précédente ainsi que l'état du contexte de la fonction à chaque appel.

	T	*I*	*j*	
recurseRec(T,1)	*@A*	*1*	*0*	
recurseRec(T,i+1)	*@A,@A*	*1,2*	*0,1*	
recurseRec(T,i+1)	*@A,@A,@A*	*1,2,3*	*0,1,0*	
recurseRec(T,i+1)	*@A,@A,@A,@A*	*1,2,3,4*	*0,1,0,1*	
-> 1	*@A,@A,@A*	*1,2,3*	*0,1,0*	
->1	*@A,@A*	*1,2*	*0,1*	
-> 2	*@A*	*1*	*0*	
-> rien				

3. Le tableau est passé par référence. S'il était passé par valeur, quel serait l'effet sur la complexité ?
Le tableau serait alors recopié à chaque appel récursif. Au niveau de la complexité en temps, cela ajoutera $O(N^2)$ opérations (N fois une copie dont la complexité est en O(N)).Au niveau de la mémoire de la même manière $O(N^2)$ entiers. Donc, l'algorithme qui est dans l'état actuel
   - *en O(N) en temps passerait en $O(N^2)$,*
   - *en O(1) en mémoire passerait en $O(N^2)$.*

   La complexité serait donc très dégradée

Question 4. Soit un tableau B d'entiers de taille N.

1. Ecrire la fonction *egal* qui prend pour paramètre un index *I* et le tableau *B* et dont le résultat est *vrai* si *B[I]* a la même valeur que *B[I+1]*.

```
fonction egal(ref B :tableau[1..N] d'entiers, val i :entier) :booléen ;
   début
      si i>=N alors
            retourner (NULL)
      sinon
           retourner(B[i]==B[i+1])
      finsi
   fin
finfonction
```

2. Ecrire une fonction *compteEgal* qui utilise la fonction *egal* et dont le résultat est le nombre de fois où deux entiers consécutifs sont égaux. Par exemple si le tableau contient (4,2,2,7,5,5,5,6,2,2), le résultat est 4 ((2,2), (5,5), (5,5),(2,2)).

```
fonction compteEgal(ref B :tableau[1..N] d'entiers) :entier ;
   var cpt,i :entier ;
   début
     cpt=0 ;
     pour i allant de 1 à N-1 faire
           si egal(B,i)  alors cpt=cpt+1 finsi
     finpour
     retourner(cpt)
   fin
finfonction
```

3. Ecrire une fonction *compresse* dont le résultat est un tableau de taille M, M≤N qui contient l'ensemble des valeurs de B sans répétition. On utilisera les fonctions *compteEgal* et *egal.* Par exemple la séquence (4,2,2,7,5,5,5,6,2,2) devient (4,2,7,5,6,2).

Il n'est pas possible de dimensionner dynamiquement le tableau (dans le contexte du programme INF102) donc il faut avoir un tableau résultat en paramètre et renvoyer la taille utile (fonction compteEgal question 2).

```
fonction compresse(ref B :tableau[1..N] d'entiers, ref R :tableau[1..N] d'entiers) :entier;
   var cpt,iR,i :entier ;
   début
     cpt=N-compteEgal(B) ;iR=0 ;
     pour i allant de 1 à N-1 faire
           si egal(B,i)  alors iR=iR+1 ;R[iR]=B[i] finsi
     finpour
     si non(egal(B,N-1)) alors iR=iR+1 ;R[iR]=B[N] finsi
     retourner(cpt)
   fin
finfonction
```

<u>Remarque</u> : iR==cpt, on aurait pu écrire retourner(iR). Cependant, si on utilise un tableau dynamique, alors cpt permet de construire le tableau « à la volée ».

Question 5.

1. Donner les primitives existantes pour les listes doublement chaînées et qui n'existent pas pour les listes simplement chaînées. Pourquoi la fonction *estDernier* n'est-elle pas considérée comme une primitive ?

   précédent et dernier.

   La fonction estDernier n'est pas une primitive car elle s'exprime de façon directe en utilisant la primitive suivant.

2. Soit L une liste simplement chaînée d'entiers.

   - Que contient L et P après la séquence d'instructions suivante

```
var L :listeSC d'entier ;
var P :^entier ;
creerListe(L) ;
ajouterEnTete(3,L) ;
ajouterEnTete(12,L) ;
P=premier(L) ;
ajouterAprès(6,L,P) ;
P=suivant(L,P) ;
ajouterAprès(4,L,P) ;
P=premier(L) ;
supprimer(L,P) ;
P=suivant(L,premier(L)) ;
```

   P=*adresse de 4=@4*……

   L=*(6,4,3)*... ……

   - Ecrire une fonction *compteEgalListe* dont le résultat est le nombre de fois où deux entiers consécutifs dans la liste L sont égaux. On pourra s'inspirer de la question 4.

```
fonction compteEgalListe(ref L :listeSC d'entiers) :entier ;
   var compte :entier ;
   var P :^entier ;
   fonctionEgal (ref L :listeSC d'entiers,ref P :^entier) :booléen ;
      début
        retourner(contenu(L,P)== contenu(suivant(L,P))
      fin
   finfonction
   début
      si non(listeVide(L)) alors
                P=premier(L) ;
              tant que non(suivant(L,P)==NIL) faire
                 si egal(L,P) alors
                    compte=compte+1 ;
                 finsi
              fintantque
      finsi
      retourner(compte) ;
   fin
finfonction
```

Version récursive

```
fonction compteEgalListe(ref L :listeSC d'entiers) :entier ;
fonction compteEgalListeRec(ref L :listeSC d'entiers,ref P :^entier) :entier ;
   var Psuiv :^entier ;
   début
     si estDernier(L,P) alors
            retourner(0) ;
    sinon
        Psuiv=suivant(L,P) ;
       si contenu(L,P)== contenu(L,Psuiv) alors
            retourner(1+compteEgalListeRec(L,Psuiv))
       sinon
           retourner(compteEgalListeRec(L,Psuiv))
     finsi
   fin
finfonction
début
   retourner(compteEgalListeRec(L,premier(L)) ;
fin
finfonction
```

FIN