Université de Technologie de Compiègne — Documents non autorisés
Examen médian NF16, Printemps 2011 — Durée 2 heures

Dans un souci d'équité, il ne sera pas répondu à des questions pendant la durée de l'épreuve (sauf aux étudiants étrangers ayant des difficultés pour comprendre le français aidés pour la compréhension du sujet). Si l'énoncé semble comporter une imprécision, faites un choix pour la lever que vous indiquerez clairement dans votre copie. Assurez-vous par une relecture attentive de l'énoncé que la réponse à votre question n'y figure pas. **Chaque partie est à rédiger sur une copie séparée, un point vous sera retranché sinon**.

Exercice I (copie numéro I, 4 points).

1- Écrire la fonction *int somme(int *tab1,int*tab2)* qui permet de retourner la somme des cases du tableau Comprises entre tab1 et tab2.
NB : Il n'est pas permis de déclarer d'autres variables.

2- Calculer la complexité de cette fonction :

```
int Calcul(int n)
{
        int i = 0;
        int j = 0;
        while (i <= n)
        {
                i = 2*j + i + 1;
                j++;
        }
        return j-1;
}
```

Exercice II (copie numéro I, 6 points).

1- Écrire une fonction en langage C qui permet d'afficher le contenu des n premières cases d'un tableau.

void Afficher(int Tab[], int n) ;

2- Écrire une fonction en langage C qui permet de partitionner une somme S sur S cases d'un tableau à partir de sa taille actuelle et retourne la nouvelle taille du tableau :

int Partitionner(int Tab[], int n, int S) ;

3- Soit l'algorithme suivant :

```
Combinaison(s :entier)
Début
        n,somme :entier
        Tableau Tab
        Faire
                n=Partitionner_alg(n,som,Tab)
                Afficher_alg(Tab,n)
                Somme=Tab[n]-1
```

```
            n=n-1
            Si(n>0)
                  Tab[n]= Tab[n]+1 ;
      Tan que (n>0)
Fin
```

Afficher_Alg et *Partitionner_Alg* sont les versions algorithmiques des deux fonctions *Afficher* et *Partitionner*.

a- Exécuter à la main l'algorithme précédent avec S=4.
b- Que fait cet algorithme
c- Traduire l'algorithme en langage C
d- Calculer sa complexité

4- Écrire en langage C une version récursive de l'algorithme Combinaison

---

Exercice 4 (copie numéro II : 10 points). ***Toutes les procédures dans cet exercice seront écrites en C.***

On cherche à développer un outil informatique pour la gestion des notes des étudiants d'une université. Chaque étudiant est représenté par un identificateur unique du type entier.

1. Ecrire la structure *notes* qui contient les 3 champs : *matiere, note et suivant* qui ont comme types respectivement chaîne de caractère, réelle et pointeur sur une structure *notes*. Définir le type *listeNotes*, pointeur sur la structure *notes*. *listeNotes* servira à définir des listes de notes obtenues pour un étudiant donné.
2. On veut définir une liste d'étudiants qui ont déjà passé des épreuves. Pour cela on définit la structure suivante :
   *struct etudiant {int identificateur ; listeNotes L; int nbEpreuves ; float moyenne ; struct etudiant * suivant ;}*
   On définit aussi la liste *listeEtud* comme pointeur sur *struct etudiant*. Chaque élément de cette liste contient les champs : *identificateur* qui identifie l'étudiant, *L* une liste de matières et de notes correspondantes, *nbEpreuves* le nombre d'épreuves effectuées ou en d'autres termes le nombre d'éléments de la liste *L*, *moyenne* la moyenne sur les notes obtenues, et *suivant* un pointeur sur l'élément suivant.
   2.a. Ecrire les fonctions *nouvListeNotes* et *nouvListeEtud* qui retournent respectivement une liste vide de notes et une liste vide d'étudiants.
   2.b. Ecrire la fonction *ajoutNote* suivante: *ListeNotes ajoutNote (ListeNotes ln, char[50] mat, int note)* qui étant données une liste de type *listeNotes*, une matière *mat* et une note, ajoute une nouvelle matière et la note correspondante en tête de *ln*, et retourne la liste modifiée.
   2.c Ecrire une fonction *ajout* suivante : *listeEtud ajout (listeEtud let, int identifEtud, char[50] mat, float note)* qui étant donnée une liste d'étudiants *let* ajoute une note pour l'étudiant dont l'identificateur est *identifEtud* . Il faut faire attention aux 2 cas possibles, ou bien l'étudiant en question a déjà passé une épreuve et par conséquent il figure déjà dans *let*, ou bien il n'a passé aucune épreuve et du coup il ne figure pas dans la liste, dans le dernier cas il faut ajouter un nouvel élément correspondant à cet étudiant à *let*. Dans tous les cas, il faut mettre à jour les champs affectés par l'ajout d'une note.
3. Ecrire une procédure *subGroup* qui étant données une liste d'étudiants *(listeEtud) let* et une matière donnée, crée 3 listes d'étudiants (*listeEtud*) : *let1* liste des étudiants de *let* ayant réussi l'épreuve de cette matière, *let2* la liste des étudiants de *let* qui l'ont raté et *let3* la liste des étudiants de *let* qui n'ont pas fait l'épreuve de cette matière.
4. Ecrire la fonction *classement (listeEtud let)* qui affiche à l'écran le classement des étudiants (ordre décroissant de la moyenne).
