

Examen

Date : 18 janvier 2006
Documents et supports de cours autorisés.

Remarques :

- * Vous ferez attention à la gestion des erreurs.
- * Toutes les questions peuvent être traitées indépendamment, en supposant connues les fonctions et procédures définies dans les questions précédentes. Vous veillerez d'ailleurs pour répondre aux questions à utiliser les fonctions ou procédures des questions précédentes, si possible.
- * Vous pouvez définir des procédures ou des fonctions qui ne sont pas demandées, mais dont vous sentez qu'elles sont nécessaires pour répondre proprement à une question.

I - Première partie : listes chaînées de personnes

Nous allons considérer deux structures imbriquées :

- * une structure de données **struct date**, de type synonyme **DATE**, qui contiendra comme champs 3 entiers **jour**, **mois**, **annee**,
- * une structure de données **struct personne** (dont on définira le type synonyme **PERSONNE**) ayant pour données :
 - un pointeur sur un chaîne de caractères de nom **nom**, qui correspondra au nom de la personne,
 - un tableau **telephone** de 10 caractères, qui correspondra au numéro de téléphone de la personne,
 - une structure de données de type **DATE**, qui correspondra à la date à laquelle a été rentrée la fiche dans la liste chaînée.
 - un pointeur sur **struct personne**, nommé **suisant**, qui pointera sur l'élément suivant de la liste chaînée.

1- écrire les déclarations des structures **struct personne** et **struct date**, ainsi que leurs types synonymes.

```
typedef struct date
{
    int jour, mois, année ;
} DATE ;

typedef struct personne
{
    char *nom ;
    char telephone[10] ;
    DATE entree ;
    struct personne *suisant ;
} PERSONNE ;
```

2- écrire une fonction **AllouePersonne** qui alloue la mémoire nécessaire pour un élément de type **struct personne**.

```
PERSONNE* AllouePersonne()
{
    PERSONNE *pers=NULL ;

    if( pers=(PERSONNE*)malloc(sizeof(PERSONNE))==NULL )
    {
        printf("Memory allocation error in AllouePersonne()\n") ;
        /*exit(1) ;*/ /* pas obligatoire, selon que l'on gère les
erreurs ou non dans la suite */
    }
    else
    {
        pers->nom=NULL ;
        pers->suisant=NULL ;
    }
    return pers ;
}
```

3- écrire une procédure **SaisiePersonne** qui permet à l'utilisateur de saisir au clavier les différents champs d'un élément de type **struct personne** que l'on fera passer par adresse comme argument d'entrée.

```
DATE SaisieDate()
{
    DATE entree ;

    printf("\nQuel jour ? ") ;scanf("%d",&entree.jour) ;
    printf("\nQuel mois ? ") ;scanf("%d",&entree.mois) ;
    printf("\nQuel annee ? ") ;scanf("%d",&entree.annee) ;
    printf("\n")

    return entree ;
}

char* SaisieNom()
{
    char* nom=NULL ;
    char temp[100] ;
    int Nbchar=0, compt=0 ;

    fgets(stdin,temp,99) ;
    Nbchar=strlen(temp) ;
    nom=(char*)malloc(Nbchar*sizeof(char)) ;
    if(nom==NULL) printf("Memory Allocation Error in SaisieNom\n") ;
    else
    {
        for (compt=0 ;compt<=Nbchar ;compt++) nom[compt]=temp[compt] ;
    }
    fgets(stdin,temp,99) ; /* pour enlever le caractère « entrée » */
    return nom ;
}
```

```

void SaisiePersonne(PERSONNE *pers)
{
    int compt=0 ;
    if(pers==NULL)
        {
            printf("No Memory Allocation in SaisiePersonne()\n") ;
        }
    else
        {
            printf("\nNom ? (<100 charac.) ") ; pers->nom=SaisieNom() ;
            printf("\nTelephone ? ") ;
            for(compt=0 ;compt<10 ;compt++)
                scanf("%d",&pers->telephone[compt]) ;
            pers->entree=SaisieDate() ;
        }
}

```

4- écrire une fonction **RecherchePersonne** pour chercher une personne dans une liste chaînée selon son nom (la tête et le nom à chercher seront passés en argument d'entrée) qui renverra l'adresse de la structure correspondant au nom ou NULL si le nom n'appartient pas à la liste chaînée.

```

PERSONNE *RecherchePersonne(PERSONNE *tete, char*nom)
{
    PERSONNE *current =tete;
    while(current !=NULL && strcmp(current->nom,nom)!=0)
        {
            current=current->suivant ;
        }
    return current ;
}

```

5- écrire une fonction **InserePersonne** qui permet d'insérer une nouvelle personne (représentée par une structure de type **PERSONNE** que l'on fera passer à la fonction par adresse) dans la liste chaînée ; cette insertion devra se faire de sorte à :

- ce que la liste chaînée soit triée selon l'ordre alphabétique du champ nom
- ce qu'aucun élément ne soit en double (même nom, même numéro de téléphone) dans la liste chaînée résultante. Si une entrée a le même nom, on privilégiera l'élément qui est le plus récent.

```

int CompareDate(DATE date1, DATE date2)
{
    int diff_approx=(date1.annee-date2.annee)*372+(date1.mois-
date2.mois)*31+date1.jour-date2.jour ;

    return diff_approx ;
}

```

```

PERSONNE *InserePersonne(PERSONNE *tete, PERSONNE *pers)
{
    PERSONNE *current=tete, *prec=NULL ;
    /* les cas singuliers */
    if (tete==NULL) return pers ;
    if (pers==NULL) return tete ;
    if (strcmp(pers->nom,tete->nom)<0)
        {pers->suisant=tete ;return pers ;}
    if (strcmp(pers->nom,tete->nom)==0)
        if (CompareDate(pers->entree,tete->entree)>0)
            {pers->suisant=tete->suisant ; return pers ;}
    /* les cas courants*/
    while(current->suisant !=NULL && strcmp(pers->nom,current->nom)>0)
        { prec=current ;current=current->suisant ; }
    if (strcmp(pers->nom,current->nom)<=0)
        if (strcmp(pers->nom,current->nom)==0)
            {
                if (CompareDate(pers->entree,current->entree)>0)
                    {
                        pers->suisant=current->suisant ;
                        prec->suisant=pers ;
                        LiberePersonne(current) ; /* cf. question 7*/
                    }
            }
        else
            {
                prec->suisant=pers ;
                pers->suisant=current ;
            }
        else
            {
                current->suisant=pers ;
            }
    return tete ;
}

```

6- écrire une fonction **FusionnePersonne** qui fusionne deux listes chaînées de type **PERSONNE** dont on fera passer des pointeurs sur leurs têtes. La liste chaînée finale respectera l'ordre alphabétique sur le nom et ne contiendra aucun doublon (même nom, même adresse).

```

PERSONNE *FusionnePersonne(PERSONNE *tete1, PERSONNE *tete2)
{
    PERSONNE *current =tete2;
    /* cas triviaux */
    if (tete1==NULL) return tete2 ;
    if(tete2==NULL) return tete1 ;
    /* autre cas*/
    while(tete2 !=NULL)
    {
        current=tete2 ;
        tete2=tete2->suisant ;
        tete1=InserePersonne(tete1,current) ;
    }
    return tete1 ;
}

```

7- écrire une fonction **SupprimePersonne** qui supprime une personne de la liste chaînée, dont les adresses ont été passées en argument d'entrée.

```
void LiberePersonne(PERSONNE *pers)
{
    free(pers->nom) ;
    free(pers) ;
}

PERSONNE *SupprimePersonne(PERSONNE *tete, PERSONNE *pers)
{
    PERSONNE *prec=NULL, *current=tete ;
    if (tete==NULL || pers==NULL) return tete ;
    if (tete==pers)
        tete=tete->suivant ;
    else
    {
        while(current !=NULL && current !=pers)
            {prec=current ;current=current->suivant ;}
        if(current !=NULL)
            prec->suivant=current->suivant ;
    }
    LiberePersonne(pers) ; pers=NULL ;
    return tete ;
}
```

8- écrire une fonction **SupprimeListePersonne** qui supprime l'ensemble d'une liste chaînée et libère la mémoire allouée.

```
void SupprimeListePersonne(PERSONNE *tete)
{
    while(tete !=NULL)
        {tete=SupprimePersonne(tete,tete) ;}
}
```

II - Seconde partie : un annuaire

Nous désirons gérer un annuaire de personne en définissant un tableau d'adresse où chaque case du tableau pointe sur la tête d'une liste chaînée ne contenant que des personnes dont le nom commence par la même lettre (en majuscules ou en minuscules). Pour gérer ce tableau comme une pile, nous allons considérer une structure de données **struct annuaire** (de type synonyme **ANNUAIRE**) contenant :

- un double pointeur **tab** sur des **struct personne** qui pointeront sur le tableau d'adresses des têtes des listes chaînées de type **struct personne**.
- un entier **Max** qui correspondra au nombre maximum de listes chaînées que peut contenir le tableau.
- un entier **Nb** qui représentera le nombre de listes chaînées contenues.

9- écrire la déclaration de la structure **struct annuaire**, ainsi que son type synonyme.

```
typedef struct annuaire
{
    PERSONNE **tab ;
    int Max ;
    int Nb ;
} ANNUAIRE ;
```

10- écrire une fonction **AlloueAnnuaire** qui alloue et renvoie l'adresse sur une structure de type **struct annuaire** ; cette fonction devra aussi allouer le champ **tab** de sorte à contenir au maximum 26 cases (il n'y a que 26 lettres dans l'alphabet) ainsi que donner des valeurs initiales aux autres champs.

```
ANNUAIRE *AlloueAnnuaire()
{
    ANNUAIRE *annu=NULL ;
    int compt=0 ;

    annu=(ANNUAIRE *)malloc(sizeof(ANNUAIRE)) ;
    if(annu==NULL)
    {
        printf("Memory allocation error in AlloueAnnuaire()\n");
        /* exit(1);*/
    }
    else
    {
        annu->Max=26 ;
        annu->Nb=0 ;
        annu->tab=(PERSONNE**)malloc(26*sizeof(PERSONNE*)) ;
        for(compt=0 ;compt<26 ;compt++) annu->tab[compt]=NULL ;
    }
    return annu ;
}
```

11- écrire une fonction **InserePersonnesAnnuaire** qui permet d'insérer une liste chaînée de type **struct personne** dans une structure de type **ANNUAIRE** (les deux ayant été passés par adresse en argument d'entrée). Cette fonction devra fractionner la liste chaînée si elle contient des personnes aillant des noms commençant par des lettres différentes. Nous rappelons que l'annuaire doit être géré comme une pile : si une liste chaînée correspond à une liste déjà existante dans l'annuaire, elles sont fusionnées ; sinon, la liste chaînée est ajoutée dans la pile comme un nouvel élément.

```
void InsereUnePersonneAnnuaire(PERSONNE *pers, ANNUAIRE *annu)
{
    int compt=0 ;
    int test=0 ;

    for (compt=0 ;compt<annu->Nb ;compt++)
        if (pers->nom[0]==annu->tab[compt]->nom[0])
        {
            InserePersonne(annu->tab[compt],pers) ;
        }
```

```

        test=1 ;
    }
    if (test==0
    {
        if (annu->Nb==annu->Max)
        {
            printf("FIFO Error in InsereUnePersonneAnnuaire()\n");
        }
        else
        {
            InserePersonne(annu->tab[annu->Nb],pers) ;
            annu->Nb++ ;
        }
    }
}

```

```

void InserePersonnesAnnuaire(PERSONNE *pers, ANNUAIRE *annu)
{
    PERSONNE *current=pers, *tete=pers ;

    while(tete != NULL)
    {
        current=tete ;
        tete=tete->suivant ;
        InsereUnePersonneAnnuaire(current,annu) ;
    }
}

```

12- écrire une fonction **RecherchePersonneAnnuaire** qui permet de rechercher une personne dans l'annuaire. A partir du nom passé en argument d'entrée, elle devra renvoyer un pointeur sur la fiche correspondant à la personne ou la valeur NULL si ce nom n'appartient pas à l'annuaire.

```

PERSONNE *RecherchePersonneAnnuaire(ANNUAIRE *annu, char *nom)
{
    int compt=0 ; int test=0;
    PERSONNE *pers=NULL;
    if (annu==NULL || nom==NULL) return NULL ;

    for(compt=0;compt<annu->Nb;compt++)
        if(nom[0]==annu->tab[compt]->nom[0])
        {
            pers= SupprimePersonne(annu->tab[compt],nom) ;
            return pers ;
        }
    return NULL ;
}

```

13- écrire une fonction **SupprimerPersonneAnnuaire** qui permet de supprimer une personne dans l'annuaire.

```
void SupprimerPersonneAnnuaire(ANNUAIRE *annu, PERSONNE *pers)
{
    int compt=0, int decal=0 ;

    for(compt=0;compt<annu->Nb;compt++)
        if (nom[0]==annu->tab[compt]->nom[0])
            {
                annu->tab[compt]= SupprimerPersonne(annu->tab[compt],pers) ;
                if (annu->tab[compt]==NULL)
                    {
                        /* decalage */
                        annu->Nb--;
                        for(decal=compt;decal<annu->Nb;decal++)
                            annu->tab[decal]=annu->tab[decal+1];
                        annu->tab[annu->Nb]=NULL;
                    }
                break;
            }
}
```

14- écrire une fonction **FusionneAnnuaire** qui permet de fusionner 2 annuaires en un seul, passés en arguments d'entrée.

```
/* La fusion est réalisée dans l'annuaire
passé en premier argument */
void FusionneAnnuaire(ANNUAIRE *annu1,ANNUAIRE *annu2)
{
    int compt=0 ;
    for(compt=0;compt<annu2->Nb;compt++)
        InserePersonnesAnnuaire(annu2->tab[compt],annu1) ;
}
```

15- écrire une fonction **SupprimeAnnuaire** qui permet d'effacer le contenu d'un annuaire, ainsi que de libérer la mémoire allouée.

```
void SupprimeAnnuaire(ANNUAIRE *annu)
{
    int compt=0 ;
    for(compt=0;compt<annu->Nb;compt++)
        SupprimeListePersonne(annu->tab[compt]) ;
    free(annu->tab) ;
    free(annu) ;
}
```