Groupe :	Nom : **correction**	Prénom :	Note : / 39

Contrôle de langage C – 17 mai 2008

Durée 2h – **Document autorisé : 1 feuille double personnelle de notes**

Remarques
- dans ce sujet, on ne se soucie pas d'inclure les fichiers (`#include …`)
- les questions et sous-questions sont indépendantes

On veut gérer des sommets de montagnes. Un sommet est stocké dans une structure qui contient les informations suivantes :
1. le nom du sommet
2. l'altitude (en mètres) du sommet
3. le massif dont le sommet fait partie (Vercors, Chartreuse, Belledonne, etc)

La structure, ainsi que le nouveau type associé et quelques constantes, sont déclarés comme suit :

```
struct sommet {
  char nom[200]; // nom du sommet
  int altitude; // altitude du sommet
  char massif[200]; // nom du massif
};
typedef struct sommet t_sommet;

#define OK 0       // à utiliser
#define ERREUR 1   // si nécessaire
#define FIN -1
```

- On rappelle que les notations `char* tab` et `char tab[]` sont équivalentes dès lors que le tableau d'adresse `tab` existe en mémoire.

Question 1

Ecrire la fonction `void AfficherSommet(t_sommet som)` qui affiche à l'écran de la façon la plus simple qui soit le nom du sommet, son altitude et le massif dont il fait partie.

```
void AfficherSommet(t_sommet som)
{

printf("%s %d %s\n", som.nom, som.altitude, som.massif);


1 pour l'utilisation de printf et le bon nombre de paramètres (c-à-d 3)
1 pour %d et %s (0 si seulement %s ou %d correct)
1 pour la syntaxe structure.membre (0 si pas utilisée correctement pour tous
les params)

}
```

Question 2

Ecrire la fonction `int AltitudeMax(t_sommet tabsom[], int nb)` qui renvoie l'altitude du sommet le plus élevé, parmi les sommets du tableau `tabsom` de taille `nb`.

```
int AltitudeMax(t_sommet tabsom[], int nb)
{

int i;
int max = 0;
for (i = 0; i < nb; i++)
{
if (tabsom[i].altitude > max)
max = tabsom[i].altitude;
}
return max;


1 pour commencer à 0
1 pour traiter le bon nombre d'éléments
1 pour la notation tabsom[i].altitude ou *(tabsom + i).altitude
1 pour l'utilisation correcte de return
1 pour retourner un résultat correct


}
```

Question 3

a) Ecrire la fonction `t_sommet CreerSommet()` qui crée dynamiquement une variable structurée de type `t_sommet` et renvoie son adresse ou `NULL` si la création n'a pas été possible. La fonction n'initialise pas les membres.

```
t_sommet * CreerSommet()
{
return (t_sommet *) malloc(sizeof(t_sommet));


1 pour l'utilisation de malloc()
1 pour l'utilisation correcte de sizeof
OK si pas de cast (il n'est plus recommandé de caster le retour de malloc)









}
```

b) Dans un main, à l'aide de la fonction `CreerSommet`, créer un nouveau sommet puis l'initialiser avec les valeurs "`Chamechaude`", "`2082`", "`Chartreuse`". Afficher le sommet avec la fonction `AfficherSommet`, puis libérer la mémoire allouée avant de sortir du programme.

```
int main()
{
t_sommet* som;
som = CreerSommet();
if (som)
{
strcpy(som->nom, "Chamechaude");
som->altitude = 2082;
strcpy(som->massif, "Chartreuse");
AfficherSommet(*som);
free(som);
}
else printf(" création sommet impossible\n ") ;

return EXIT_SUCCESS ;


1 pour la déclaration de t_sommet*
1 pour l'appel correct à CreerSommet et l'affectation
1 pour le test de la validité de som
1 pour l'utilisation de -> ou (*som).
1 pour l'utilisation strcpy (ou strncpy)
1 pour le passage de *som à AfficherSommet
1 pour l'appel correct à free
1 pour un affichage de message d'erreur
1 pour return EXIT_SUCCESS ou return 0


}
```

Question 4

Ecrire la macro `DENIVELE(sommet1, sommet2)` qui donne la différence d'altitude entre les deux sommets sommet1 et sommet2

```
#define DENIVELE(sommet1, sommet2) (sommet1.altitude - sommet2.altitude)

1 pour la syntaxe correcte #define DENIVELE(sommet1, sommet2)
1 pour le calcul (sommet1.altitude - sommet2.altitude)
1 si les parenthèses sont présentes autour de l'expression calculée
```

Question 5

On veut normaliser les noms de sommets et de massifs, en mettant leur première lettre en majuscule, et en minuscule les autres lettres.

a) Ecrire la fonction `void Normaliser1(char* ch)`, qui, **à l'aide d'un masque de bits et d'un opérateur logique**, **sans appeler de fonction**, met en majuscule la première lettre de la chaîne, et en minuscules les lettres suivantes ("`MoNT BLAnc`" devient "`Mont blanc`"). On rappelle qu'en ASCII, le bit 5 différencie les lettres minuscules (..1. ....) des majuscules (..0. ....).

```
void Normaliser1(char* ch)
{
int i = 1;

// on met a zero le bit 5 des premieres lettres

if (ch[0] >= 'a' && ch[0] <= 'z')
{
ch[0] = ch[0] & 0xDF; // OU <1101 1111>2, <223>10
}

// on met a 1 le bit 5 des autres lettres du nom

while (ch[i] != 0)  // ou bien != '\0'
{
if (ch[i] >= 'A' && ch[i] <= 'Z')
ch[i] = ch[i] | 0x20 ; // OU <0010 0000>2, <32>10
i++ ;
}


1 pour traiter tous les caractères (ie l'arrêt au caractère NULL)
1 pour la comparaison à 'A'/'Z'/'a'/'z'
1 pour l'utilisation du ET
1 pour l'utilisation du OU
1 pour la bonne valeur du masque majuscule
1 pour la bonne valeur du masque minuscule

}
```

b) Ecrire la fonction `void Normaliser2(char* ch)`, toujours **à l'aide d'un masque de bits et d'un opérateur logique**, **sans appeler de fonction**. On veut maintenant mettre en majuscule le premier caractère, ainsi que ceux qui suivent un espace (" "), une apostrophe (') ou un tiret (-). Les autres caractères doivent être mis en minuscule. Exemple: "`aGuille dE l'argentiere`" devient "`Aiguille De L'Argentiere`".

```
void Normaliser2(char* ch)
{
int i = 0;
int maj = 1;
while (ch[i] != 0) {
if (maj && ch[0] >= 'a' && ch[0] <= 'z')
{
ch[0] = ch[0] & 0xDF; // OU <1101 1111>2, <223>10
} else if (!maj && ch[i] >= 'A' && ch[i] <= 'Z')
ch[i] = ch[i] | 0x20 ; // OU <0010 0000>2, <32>10
}
maj = (ch[i] == ' ' || ch[i] == '-' || ch[i]='\'');
i++ ;
}

1 pour traiter tous les caractères (ie l'arrêt au caractère NULL)
1 pour échappement correct de l'apostrophe
1 pour donner un résultat correct
}
```

Question 6

a) Ecrire la fonction `void EcrireSommet(t_sommet* som, FILE* f)` qui écrit le contenu du sommet pointé par `som` dans `f` (déjà ouvert en écriture) avec la fonction `fwrite`. On ne demande pas de tester la valeur de retour de `fwrite`.

```
void EcrireSommet(t_sommet* som, FILE* f)
{
fwrite(som, sizeof(*som), 1, f);


1 pour fwrite avec les bons parametres

}
```

b) Ecrire la fonction `int LireSommet(FILE* f, t_sommet* som)` qui lit la structure située sous la tête de lecture de `f` (déjà ouvert en lecture) et la range dans la structure pointée par `som` avec la fonction `fread`. Cette fonction retourne la constante `FIN` s'il n'y a rien à lire (fin de fichier) ou `OK` sinon.

```
int LireSommet(FILE* f, t_sommet* som)
{
int rep = fread(som, sizeof(*som), 1, f) ;
if (rep == 0)
return FIN ;
else
return OK ;



1 pour la bonne gestion des retours

}
```

c) Ecrire la fonction `int main(...)` qui récupère comme argument passé en ligne de commande un nom de fichier. Ce fichier contient plusieurs sommets écrits avec `EcrireSommet`. La fonction `main` lit à l'aide de `LireSommet` tous les sommets contenus dans le fichier, normalise le nom du sommet et du massif avec `Normalise2`, et les affiche avec `AfficherSommet`.
Voici un exemple d'exécution :

```
bertolin@st-dg-01:~/c$ prog fichier.bin
```

```
int main (int argc, char* argv[])
{

t_sommet som ;
int rep ;
FILE* f ;


if (argc != 2)
{
printf("pas le bon nombre d'arguments\n") ;
return EXIT_FAILURE ;
}

f = fopen(argv[1], "rb") ;
```

```
if (!f)
{
printf("pb a l'ouverture\n") ;
return EXIT_FAILURE ;
}


rep = LireSommet(f, &som) ;
while (rep == OK)
{
Normalise2(som.nom);
Normalise2(som.massif);
AfficherSommet(som) ;
rep = LireSommet(f, &som) ;

}

fclose(f) ;

return EXIT_SUCCESS ;




1 pour test du nombre d'arguments
1 pour déclaration FILE*
1 pour appel fopen parfait (ok si "r" à la place de "rb": c'est pareil si le
système est POSIX)
1 pour test d'ouverture
1 si appel correct de Normalise2 pour traiter le nom et le massif
1 pour fclose

 }
```