Programmation C
Examen 6 janvier 2020 durée 2h

Documents autorisés et consignes

Documents autorisés :

- Mémento de fonctions du langage C distribué en cours.
- Trois feuilles numérotées A4 recto-verso de vos notes personnelles. Chaque feuille doit porter votre nom.

Votre code doit être écrit de façon lisible, avec des indentations et des accolades appropriées permettant de voir la fin de blocs de code (fin de boucles, etc.). Il est inutile d'écrire les `#include`. Si les fonctions de la bibliothèque standard, comme `malloc`, échouent vous pouvez terminer immédiatement avec `exit()`.

1 Chaînes de caractères

Dans ce qui suit « chaîne de caractère » signifie toujours une chaîne de caractère dans le sens du langage C, c'est-à-dire une suite de caractères qui termine par le caractère que vous devez connaître.

Exercice 1 :

Pour chaque `printf` dans le programme suivant, écrire ce que ce `printf` affiche.

```c
int main(void){
  char phrase[] = "aliceaunchienetunchat";

  printf("phrase==<<%s>>\n", phrase);
  printf("<<%s>>\n", &phrase[5]);

  size_t len = strlen( phrase + 6 );
  printf("len=%u\n", (unsigned) len);

  phrase[8]='\0';
  printf("phrase=<<%s>>\n", phrase);
  len=strlen(phrase);
  printf("len=%u\n", (unsigned) len);
}
```

Exercice 2 :

Écrire la fonction

```c
    char *palindrome(const char *s)

```

qui, à partir de la chaîne de caractères `s`, construit et retourne une nouvelle chaîne de caractères obtenue par la concaténation de `s` avec son image miroir. Par exemple si `s` est `"anemarie"` alors `palindrome(s)` retournera la chaîne de caractères

`"anemarieeiramena"`

Exercice 3 :

Écrire la fonction

```
1    char *motLePlusGrand(char *s)
2
```

qui prend comme argument une chaîne de caractères `s` qui contient uniquement de lettres (reconnues grâce à la fonction `isalpha`) et espaces (reconnues comme espaces par la fonction `isspace`).
On définit un mot dans `s` comme une plus longue suite de lettres (sans espaces). Par exemple la chaîne de caractères `"Ala␣ma␣psa␣i␣kota␣␣"` contient cinq mots : `ala` , `ma`, `psa`, `i`, `kota`. La fonction doit retourner le mot le plus grand dans l'ordre lexicographique parmi les mots dans `s`. Par exemple pour la chaîne `s` ci-dessus la fonction retournera la chaîne de caractères `"psa"`.
Indication : `strcmp`

2 Pointeurs et tableaux

Exercice 4 :

Dans le programme suivant, pour chaque `printf`, sauf le dernier, écrire ce que ce `printf` affiche.

```
1  int main(void){
2    double t[]={0.5, -1.5, 2.2, -3.3, 4.4, -5.5, 6.6, -7.7, 8.8, -9.9, 10.0, -11.1
       };
3
4    double *pa = &t[2];
5    double *pb = &t[8];
6    ptrdiff_t d = pa - pb;
7    printf("d=%ld\n" , (long) d);
8
9    double *v = &t[6];
10   printf("%4.1f\n" , *(v-3) );
11
12   v[-1]=101;
13   v[2] =202;
14   for(int i = -2; i < 3; i++)
15     printf("v[%d] == %3.1f\n", i, v[i]);
16
17   size_t n = ???  ;
18   printf("nombre d'elements = %u\n", (unsigned) n);
19 }
```

Indiquer comment modifier la ligne 17 en remplaçant ? ? ? par une expression appropriée pour que le `printf` de la ligne 18 affiche le nombre d'éléments du tableau `t`.

3 Arbres

Exercice 5 :

Définir le type de données `arbre` qui représente un arbre binaire dont les noeuds contiennent les entiers `int`.
Écrire une fonction récursive

```
1  int somme_feuilles(arbre b)
```

qui retourne la somme des entiers stockés dans les feuilles de l'arbre `b`.
(Rappelons, même si cela devrait être superflu, qu'une feuille est un noeud sans fils.)

4 Listes

Nous définissons les types suivants :

```
1 typedef struct element{
2   struct element *suivant;
3   char *mot;
4   unsigned int n;
5 } element;
6 typedef element *liste;
```

`liste` représente une liste chaînée dont chaque élément contient : (1) un lien vers l'élément suivant (valeur NULL s'il n'y a pas d'élément suivant), (2) une chaîne de caractères `mot` et (3) un compteur `n`.

Exercice 6 :

Dans cet exercice on suppose que la liste `l` est triés dans l'ordre lexicographique de valeurs du champ `mot`. Un mot apparaît toujours au plus une seule fois (sans répétition) dans la liste `l`. Pour chaque mot sur la liste le compteur `n` correspondant est strictement supérieur à 0.
Écrire la fonction

```
1     liste insererMot(liste l, char *word)
2
```

qui insère une nouvelle chaîne de caractères `word` dans la liste.
Plus exactement `insererMot` effectue l'action suivante :

(1) Si `word` est déjà présent sur la liste `l` la fonction `insererMot` juste incrémente le compteur `n` correspondant, il n'y aura pas de création d'un nouveau élément dans la liste.

(2) Si `word` n'apparaît pas sur la liste alors un nouveau élément contenant `word` sera inséré avec la valeur `n == 1`. L'insertion doit préserver l'ordre lexicographique des mots sur la liste.

La fonction `insererMot` retourne le pointeur vers le premier élément de la liste après l'insertion.
Si initialement la liste est vide, c'est-à-dire `l==NULL`, alors après l'insertion la fonction retourne le pointeur vers l'unique élément inséré par la fonction (dont le compteur sera `n==1`).
Indication. `strcmp`

5 Tableaux et structures

Exercice 7 :

On compile et on exécute le programme suivant :

```
1
2  void f(size_t n, int tab[]){
3    size_t len = sizeof(tab);
4    printf("len = %u\n", (unsigned) len);
5    /* d'autres instructions */
6  }
7
8  int main(void){
9    int t[]={1,2,3,4,5,6,7,8,9};
10   size_t d = sizeof(t);
11   printf( "d= %u\n", (unsigned) d);
12
13   f(9, t);
14   /* d'autres instructions */
15 }
```

Est-ce que les `printf` dans les lignes 4 et 11 affichent la même valeur ?
En fait quelle(s) information(s) affichent les deux `printf` ? Il ne s'agit pas de donner les valeurs affichées mais plutôt de dire que le `printf` affiche la taille de `???` mesurée en `???`.

Exercice 8 :

Nous définissons les types suivants :

```
1  typedef struct{
2    char *text;
3    size_t cmpt;
4  } mot;
5  typedef struct{
6    size_t len;
7    size_t nbu;
8    mot *tabmots;
9  } *tabdyn;
```

`tabmots` est un pointeur vers le premier élément de tableau de `mot`.
`len` donne le nombre d'éléments de `tabmots`.
`nbu` est le nombre d'éléments « utiles » de `tabmots`, seulement les éléments dont l'indice est inférieur à `nbu` sont considérés utiles.
Chaque `mot` contient une chaîne de caractères dans le sens du C et un compteur `cmpt`.
Ecrire la fonction :

```
1    int delete_text( tabdyn t, char *txt)
```

La fonction cherche dans le tableau `t -> tabmots` un élément **utile** dont le champ `text` contient la chaîne de caractère `txt` (c'est-à-dire égale dans le sens de `strcmp`).
Si la fonction ne trouve pas d'élément recherché elle retourne `-1`.
Si la fonction trouve que i-ème élément du tableau satisfait le critère de recherche elle décrémente le compteur `cmp` associé.

Il y a deux cas :

(1) Si après la décrémentation le compteur reste positif la fonction retourne la valeur du compteur.

(2) Si après la décrémentation le compteur est 0, vous devez supprimer i-élément de `tabmots` en décalant tous les éléments utiles à partir de l'élément $i+1$ ($i+1$ vers $i$ème, $i+2$ vers $i+1$ etc.)

Ce décalage doit être fait avec une seule instruction (le décalage implémenté avec une boucle donnera moins de points). Vous décrémentez également la valeur de `nbu`.

La fonction retournera 0.

Le dessin suivant montre la configuration avec un tableau de 4 éléments dont 3 sont utiles. Dans cette situation `delete_text(t,"Alice")` diminue la valeur `cmpt` pour le mot `"Alice"`. Par contre `delete_text(t,"cat")` supprime complètement le mot `"cat"`, le nombre d'éléments utiles passera à 2.