Université Paris Diderot – Paris 7

L3 Informatique, C et Systèmes.

19 juin 2018

Durée : 3 heures.

Tous les documents papier autorisés. **Les téléphones portables interdits.**

Le sujet comporte 5 pages.

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.). Vous pouvez, si vous le trouvez utile, d'écrire de fonctions auxiliaires.*

Il est inutile d'écrire les includes. Par contre une gestion d'erreurs, même minimaliste, sera appréciée (mais ce n'est pas le point très important).

Le barème est donné à titre indicatif et peut être modifié.

QUESTIONS

Question 1: [2 points] Qu'est-ce que affiche chaque `printf` dans le programme suivant ?

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stddef.h>
4
5  int main(void){
6    char *s = "abcdefghijklmnopq";
7    char *d;
8    int i,j;
9    i=4;
10   printf("A=%c\n", *(s+i) );
11   printf("B=%c\n", s[i++] );
12   j=6;
13   printf("C=%c\n", s[++j] );
14   printf("longueur s=%lu\n", (unsigned long)strlen(s) );
15   d=s+7;
16   printf("longueur d=%lu\n", (unsigned long)strlen(d) );
17   printf("D=%c\n", d[-3] );
18
19   int tab[]={21,32,43,54,65,76,89,90,100,101};
20   ptrdiff_t g = &tab[7] - &tab[1];
21   printf("g=%ld\n", (long) g);
22   printf("h=%lu\n", sizeof(tab)/sizeof(tab[0]) );
23
24   int *pi = &tab[8];
25   printf("expr = %d\n",  *(pi-5) + 5 );
26   return 1;
27 }
```

Problème

Le but des exercices dans cette section est d'implémenter un détecteur de plagiat rudimentaire.

L'exercice i peut dépendre des exercices précédents, c'est-à-dire son l'implementation peut utiliser les fonction des exercices précédents. Cependant cela ne veut pas dire que pour écrire $i$ème exercice vous devez faire les exercices précédents, par contre vous avez droit à utiliser toutes les fonctions des exercices précédentes.

Exercice 1 [1.5 point] Écrire la fonction

```
char *supprimer_blancs(const char *s)
```

qui construit et retourne une chaîne de caractères obtenue en supprimant les caractères blancs de la chaîne `s`. Le caractère « blanc » est un caractère désigné comme « blanc » par la fonction `int isspace(int c)` du C standard. Cette fonction prend comme un argument un caractère et retourne 1 si ce caractère est un caractère « blanc » et 0 sinon. La fonction `supprimer_blanc` doit construire une nouvelle chaîne sans que `s` soit modifié.

Si `s` ne contient que des caractères blancs `supprimer_blancs` retournera `NULL` (et non pas une chaîne vide).

La mémoire allouée pour la chaîne de caractères retournée par la fonction doit correspondre exactement à ce qui est nécessaire. Une solution qui consiste à allouer beaucoup de mémoire pour y stocker une petite chaîne de caractères est prohibée.

Le but des exercices suivants est de lire un fichier texte de taille quelconque et, après la suppression de caractères blancs et lignes vides (une ligne vide est une ligne composée uniquement de caractères blancs) et de stocker le contenu du fichier ligne après ligne dans un tableau de chaînes de caractères.

Nous allons utiliser la structure de donnée suivante qui implémente un tableau dynamique de chaînes de caractères :

```
struct tablignes{
   unsigned int capacite;
   unsigned int libre;
   char **tab;
};
typedef struct tablignes *tablignes_p;
```

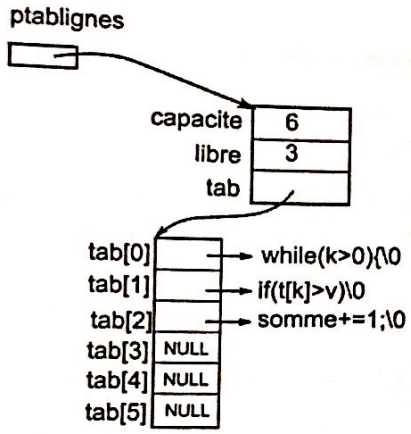
Dans la structure `struct tablignes` le champ `tab` est un pointeur vers le premier élément d'un tableau de chaînes de caractères.

Le champ `capacite` donne le nombre d'éléments dans le tableau `tab` (le nombre maximal de chaîne à stocker).

Le tableau `tab` peut être partiellement rempli, seulement les entrées `tab[0],...,tab[libre-1]` sont valables. Le champ `libre` est l'indice du premier éléments non-utilisé dans `tab`.

`libre` peut prendre les valeurs entre 0 et `capacite`, en particulier si `libre` est égal à `capacite` alors le tableau est plein.

La figure ci-dessous



présente un tableau de capacité 6 contenant actuellement 3 chaînes de caractères : `tab[0]` contient `"while(k>0){"`, `tab[1]` contient `"if(t[k]>v)"`, et `tab[2]` contient `"somme+=1;"`.

Tous les éléments de `tab` qui ne contiennent pas de chaînes de caractères doivent être mis à NULL et si un élément est NULL alors tous les éléments qui suivent doivent être NULL (pas de trous dans `tab`).

Exercice 2 [1.5 point] Écrire la fonction

```
tablignes_p create_tablignes( unsigned int capacite_init )
```

qui crée la structure de données décrite précédemment. La fonction doit créer aussi bien la structure que le tableau de chaînes de caractères dont la capacité initiale est donnée par le paramètre de la fonction. Initialement `libre` vaut bien sûr 0 et le tableau `tab` ne contient que les valeurs NULL.

Exercice 3 [1 point] Écrire la fonction

```
void detruire_tablignes(tablignes_p t)
```

qui supprime le tableau de lignes en libérant la mémoire. On suppose que la mémoire utilisée par les chaînes de caractères stockées dans `tab` a été allouée par des fonctions de la famille `malloc()` et `detruire_tablignes` doit aussi libérer la mémoire utilisée par ces chaînes de caractères.

Exercice 4 [1 point] Écrire la fonction

```
char  *get_ligne(tablignes_p t, unsigned int i)
```

qui retourne i-ème chaîne de caractères stockée dans `tab` ou NULL si `tab` ne contient pas i-ème chaîne.

Exercice 5 [2.5pt] Écrire la fonction

```
char  *ajouter_ligne_simple(tablignes_p t, char *s)
```

qui ajoute la chaîne `s` à la fin de tableau. On suppose que `s` ne possède plus de caractères blancs donc pas la peine d'appliquer `supprimer_blancs()`. La nouvelle chaîne sera ajoutée à la position indiquée par le champ `libre`. S'il n'y a plus de place pour ajouter `s` alors la fonction retournera NULL, en cas d'insertion réussie la fonction retournera son argument `s`.

Exercice 6 [3 points] Écrire la fonction

```
char *ajouter_ligne(tablignes_p t, char *s)
```

qui ajoute `s` dans `t`. Comme la fonction précédente, `s` est ajouté à la première place libre. Par contre maintenant, s'il n'y a plus de place libre il faut doubler la capacité en doublant la mémoire de `tab`.

La fonction `ajouter_ligne` retourne `s` en cas d'insertion réussie et NULL si l'insertion échoue (par exemple si on n'a pas réussi à augmenter la mémoire).

Exercice 7 [2.5 points] Écrire une fonction

```
char  *supprimer_ligne( tablignes_p t, unsigned int i )
```

qui supprime i-ème chaîne de tableau de chaînes. La fonction retourne la chaîne de caractères supprimée ou NULL s'il n'y a pas de i-ème éléments dans `tab`.

Le tableau ne doit pas contenir de trous, donc la suppression de la i-ème chaîne doit faire décaler les chaînes suivantes.

Exercice 8 [2 points] Écrire la fonction

```
tablignes_p  file2tablignes(const char *file)
```

qui lit le fichier texte `file` et construit et retourne le tableau de lignes contenant toutes les lignes de `file`.

On suppose que la taille d'une ligne du fichier `file` est au maximum `TAILLEL`, où `TAILLEL` une macro-constante avec une valeur raisonnable[1] que vous devez déclarer et utiliser dans la fonction.

La fonction lit le fichier ligne par ligne, supprime de chaque ligne les caractères blancs à l'aide de `supprimer_blancs()` et si la ligne contient des caractères qui ne sont pas des blancs alors elle ajoute la ligne (sans des caractères blancs) dans le tableau de lignes.

En cas de problème la fonction retourne NULL.

Exercice 9 [1.5 pt] Écrire une commande **(c'est-à-dire un programme qu'on exécutera depuis un terminal)**

```
similaire fa fb
```

qui, pour deux fichiers texte `fa` et `fb`, construit deux tableau de lignes à l'aide de la fonction `file2tablignes` et ensuite, en comparant à l'aide de `strcmp()`, chaque ligne du premier tableau de lignes avec chaque ligne du deuxième tableau de lignes `similaire` trouve les nombres de couples de lignes identiques dans le deux fichier.

Le programme affichera sur la sortie standard

1. le nom du fichier `fa` avec le nombre de lignes non-blancs du fichier `fa`,
2. le nom du fichier `fb` avec le nombre de lignes non-blancs du fichier `fb`,

1. 512 au moins

3. le nombre de lignes identiques[2] dans les deux fichiers.

Si pas de problèmes le programme terminera avec `exit(0)`, dans le cas contraire faire `exit` avec une valeur positive.

Exercice 10 [2.5 pt] Écrire une commande[3]

```
chercher_similaires f1 f2 ...  fn
```

qui prend comme paramètres des noms de fichiers et qui pour chaque couple de fichiers différents[4] on exécutera la commande `similaire fi fj` de l'exercice précédent. Toutes les commandes `similaire` doivent être exécutées en parallèle :

pour chaque couple `fi`, `fj`, `i< j`, `chercher_similaires` lancera un nouveau processus qui exécutera le programme `similaire fi fj`.

`chercher_similaires` termine avec `exit(0)` si toutes les commandes `similaire fi fj` ont terminé avec `exit(0)`. Dans le cas contraire `chercher_similaires` terminera avec `exit(1)`.

---

2. identiques une fois les caractères blancs supprimés
3. donc, comme dans l'exercice précédent, un programme qu'on exécutera depuis un terminal
4. différents dans le sens que on lancera `similaire fi fj` seulement pour $i < j$.