Langage C
Examen le 27 mai 2021, durée 2h30

Documents autorisés

Quatre feuilles A4 recto-verso signées de notes personnelles. Mémento de fonctions C.

Le sujet comporte 5 pages. Le barème est donné à titre indicatif et peut être modifié.

Tâchez d'écrire 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`. Dans touts les exercices vous pouvez utiliser toutes les fonctions du C standard (toutes les fonctions utilisées en cours sont des fonctions du C standard).

Exercice 1 : (3 points)

Qu'est-ce que affiche chaque `printf` du programme suivant :

```c
#include <stdio.h>
#include <string.h>
int main(void){

  char *s="abcdefghijkl";
  int i = 4 ;
  printf("A=%c\n", *(s+i) );
  printf("B=%c\n", s[++i] );

  char *c = s + 5;
  printf("longueur c=%zd\n", strlen(c) );

  printf("c decale = %c\n", c[-2] );

  int tab[]={1,2,3,4,5,6,7,8,9};
  int *pt = &tab[ 7 ];
  int *pu = &tab[ 1 ];

  printf( "diff = %td\n",  pt - pu );

}
```

Exercice 2 : 1 point

Soit

```
typedef struct elem elem;
struct elem{
  elem *next;
  int data;
};
```

Soit `elem *l;` un pointeur vers le premier élément d'une liste simplement chaînée. C'est une liste non-circulaire, la valeur du champ `next` du dernier élément de la liste est `NULL`.
Tous les éléments de la liste `l`, y compris le premier, contiennent des entiers. Le fragment de code suivant est sensé de calculer la somme de tous les entiers stockés dans la liste `l` :

```
  int somme;
  elem *cur;

  for( somme = ???, cur = ??? ;  ??? ; somme += ???,  cur = ??? )
    ; /* fin de boucle */

```

Par exemple si la liste `l` contient trois éléments avec les valeurs `data` $1, 3, 5$ respectivement alors juste après la boucle on devait obtenir `somme == 9`.
On suppose que juste avant la boucle les variables `somme` et `cur` ne sont pas initialisées.
En remplaçant les occurrences de `???` par votre code, réécrire ce fragment pour réaliser la tâche demandée. (Vous devez juste remplacer `???` par votre code, toute autre modification rapporte moins de points).

1 Listes chaînées de points

Exercice 3 : (1 point)

On définit

```
  typedef struct point point;
  struct point{
    double x;
    double y;
  }
```

Écrire la fonction

```
  double distance(const point *p, const point *q)
```

qui calcule et retourne la distance entre les points avec les adresses `p` et `q`.
Pour ceux/celles qui ont tout oublié en math, la distance entre deux points dont les coordonnées sont (x_1, y_1) et (x_2, y_2) est $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ où $a^2 = a \cdot a$. Et `sqrt` est la fonction du C qui calcule la racine carré.

Exercice 4 : (3 points)

On définit

```
typedef struct node node;

struct node{
    node *next;
    point p;
};
```

où `point` a été défini dans l'exercice précédent. `node` représente un élément d'une liste chaînées de points.
Écrire la fonction

```
  double perimetre( node *lpoint )
```

qui calcule le périmètre d'un polygone dont les sommets appartiennent à `lpoint` et où `lpoint` pointe sur le premier élément d'une liste de sommets d'un polygone. Le champ `next` du dernier élément sur la liste est `NULL` (la liste n'est pas cyclique). Les sommets sur la liste apparaissant dans l'ordre de sommets du polygone. **N'oubliez pas d'ajouter la longueur entre le premier et le dernier sommets de la liste.** Pour calculer la longueur d'un côté du polygone vous devez utiliser la fonction `distance` de l'exercice précédent (même si vous n'avez pas fait cet exercice).
On suppose que la liste `lpoint` contient au moins 3 sommets (vous n'êtes pas obligé de le vérifier, on suppose que le paramètre `lpoint` satisfait toujours cette condition).

Exercice 5 : (4 points)

Ecrire la fonction

```
  node *ajouter_point( node *lpoint, point p, int pos )
```

qui ajoute un nouveau point sur la liste `lpoint` à la position `pos`. Si `pos == 0` alors on ajoute au début de la liste, si `pos > 0` on ajoute après le `pos`-ème élément, et si

```
pos  > le nombre d'éléments de la liste
```

alors on ajoute à la fin de la liste.
La fonction retourne le pointeur sur le premier élément de la liste.
Indications. Si `lpoint == NULL` alors on ajoute le premier point sur une liste vide.
Si `pos==0` on ajoute le nouveau élément au début de la liste et la valeur retournée sera différente de `lpoint`. Ce cas est assez différent de l'ajout au milieu de la liste.

2 Chaînes de caractères

Exercice 6 : (1 point)

Une ligne d'un fichier csv est une chaîne de caractères composée de champs. Le dernier caractère dans la ligne, celui juste avant `'\0'`, est toujours le caractère de nouvelle ligne `'\n'` et il n'y a pas d'autres occurrences de `'\n'` dans la ligne.
Les champs sont séparés par un caractère délimiteur qu'on notera `del`, qui est toujours différent de caractère nouvelle ligne `'\n'`.

Aucun champ ne contient le caractère `del` qui sert à séparer les champs. Le dernier champ se termine par le caractère `'\n'` de nouvelle ligne (attention, le caractère `'\n'` *termine* le dernier champ mais n'en fait pas partie, et soulignons encore une fois, aucun champ ne contient pas de `'\n'`).
Dans les exemples suivants on suppose que `del == ';'`.

Exemples :

- `ab␣c\n` contient un seul champ qui contient `ab␣c`.
- `;ala;\n` contient trois champ, le premier et le dernier sont vides et le deuxième est composé de trois caractères `ala`.
- `bobo;;last␣\n` contient trois champ, le premier contient `bobo`, le deuxième est vide et le troisième contient `last␣`.

Écrire la fonction

```
1  char *get_field( char *line, char del, unsigned int pos )
2
```

où `line` un pointeur vers une ligne `csv` avec le délimiteur `del`. La fonction retourne le pointeur vers le premier caractère du `pos`-ème champ de la ligne `line`. Si le `pos`-ème champ est vide alors la fonction retourne le pointeur vers le caractère qui termine le champ, c'est-à-dire soit vers le caractère délimiteur soit vers le caractère de nouvelle ligne si le champ `pos` est vide et c'est le dernier champ.
On compte les champs à partir de 0, donc par exemple `get_field(line, del, 0)` retournera `line`.
Si le `pos`-ème champ n'existe pas `get_field` retournera `NULL`.

Exercice 7 : (4 points)

Écrire la fonction

```
1  char *insert_field( char *line, char del, unsigned int pos, char *champ)
2
```

où `line` est une ligne d'une fichier csv avec le caractère délimiteur `del`, `pos` est une position, et `champ` est un pointeur vers une chaîne de caractères qui ne contient ni `del` ni `'\n'`.
La fonction `insert_field` retourne une (nouvelle) ligne de fichier csv obtenue par insertion de `champ` comme le `pos`-ème champ de `ligne`.
Vous pouvez utiliser la fonction `get_field` de l'exercice précédent même si vous n'avez pas fait cet exercice.
Exemples. Dans les exemples `\n` désigne toujours le caractère de nouvelle ligne et non une suite de deux caractères.

- `insert_champ( "abc;def;ghi\n", ';', 1, "xy")` retournera le pointeur vers la chaîne de caractères `"abc;xy;def;ghi\n"`
- `insert_champ( "abc;def;ghi\n", ';', 3, "xy")` retournera le pointeur vers la chaîne de caractères `"abc;def;ghi;xy\n"`
- `insert_champ( "abc;def;ghi\n", ';', 0, "xy")` retournera le pointeur vers la chaîne de caractères `"xy;abc;def;ghi\n"`

3 Fichiers

Exercice 8 : (3 points)

Écrire un programme `pline` (la fonction `main`) telle que quand on exécute sur un terminal

```
./pline  n file
```

le programme affichera la n-ème ligne de fichier texte `file`.
Le programme doit afficher un message d'erreur (sur la sortie d'erreur `stderr` bien sûr) et terminer avec une valeur > 0 si le nombre d'arguments de la commande est incorrect.
Si `n` est trop grand (plus grand que le nombre de lignes de `file`) le programme affichera un message qui donne le nombre de lignes du fichier `file` et termine.

Pour simplifier on supposera que chaque ligne est de longueur inférieure à `LEN` où `LEN` une constante symbolique que vous devez déclarer avec la valeur de votre choix (au moins 81).
Exemple.

```
./pline 3 toto.csv
```

affichera la troisième ligne du fichier `toto.csv` (si le `toto.csv` possède au moins trois lignes).