Langage C
Devoir maison, le début le 22 juin 2020 à 8h00, durée 24h

Documents autorisés et consignes

Vous devez impérativement lire **avant de commencer les exercices** la déclaration fournie sur moodle qui précise les modalités du devoir.
Vous pouvez soit compléter la déclaration, soit faire une copie manuscrite.
Après avoir signé, vous faites une photo que vous devez joindre à votre copie sur moodle (ou envoyer par mail).

Le sujet comporte 5 pages.
Vous devez déposer sur moodle le code (un ou plusieurs fichiers `*.c`).
Si vous ne disposez pas d'ordinateur vous pouvez déposer la photo de votre copie manuscrite. Vérifiez qu'elle est lisible.
En cas de problème d'accès au moodle, vous pouvez envoyer votre copie par mail à `zielonka@irif.fr`.

1 Arbres

Nous définissons la structure `node`

```
typedef struct node node;

struct node{
  node *fg;
  node *fd;
  int term;
};
```

qui représente un sommet d'un arbre binaire. Les champs `fg` et `fd` sont de pointeurs vers les fils gauche et droit respectivement et `term` la valeur stockée dans le sommet.
Un arbre sera représenté par un pointeur vers le `node` à la racine de l'arbre.

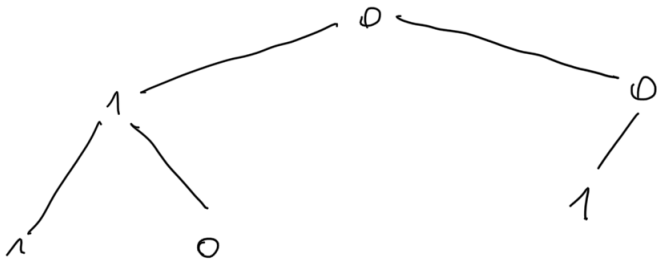
Exercice 1 :
Écrire la fonction

```
    node *dico_create( int val, node *g, node *d){

```

qui construit et retourne un arbre dont la racine stocke la valeur `val` et `g` et `d` sont des pointeurs vers le fils gauche et droit respectivement. Par exemple l'appel
`dico_create( 1, NULL, NULL )` retournera un arbre avec un seul sommet contenant la valeur `1`.

Exercice 2 :
Écrire le code (vous pouvez le mettre dans `main`) qui construit l'arbre suivant :



Les 0 et 1 dans les sommets sont les valeurs du champ `term`.

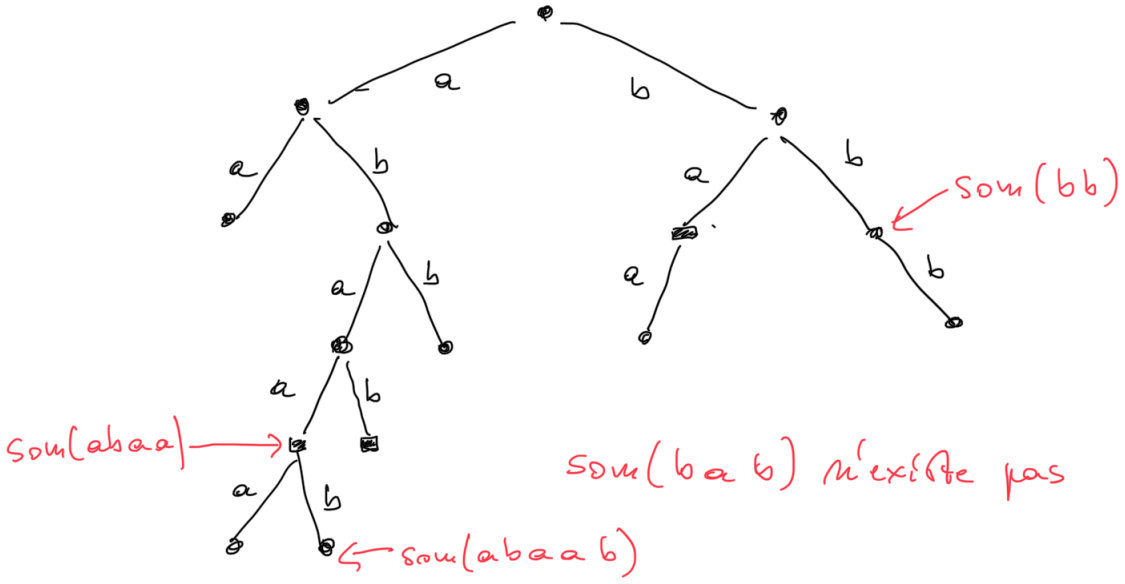
Exercice 3 :
Ecrire la fonction

```
1       int dico_card( node *a )
2
```

qui retourne le nombre de sommets avec `term == 1`.

Exercice 4 :
Un chemin de la racine vers un sommet de l'arbre peut être codé par une suite de lettres `a` et `b` où
- `a` désigne la direction « à gauche » et
- `b` désigne la direction « à droite ».

Par exemple `abaa` désigne le chemin où on va à gauche, ensuite à droite, et enfin deux fois à gauche. Par `som(abaa)` on désigne le sommet qu'on atteint à la fin de ce chemin.
On général si `w` est un mot composé de lettres `a` et `b` alors `som( w )` désigne le sommet de l'arbre binaire obtenu au bout de chemin codé par `w`.
Le dessin suivant présente `som(w)` pour certains mot `w` :



Pour un arbre `arbre` on définit `Dico(arbre)` comme l'ensemble de mot `w` sur l'alphabet de deux lettres `a` et `b` tels que le sommet `som(w)` existe et le champ `term` de ce sommet contient la valeur 1.
Par exemple pour l'arbre de l'exercice 2 `Dico(arbre)` contient trois mots `a, aa, ba`.
Le mot vide (sans aucune lettre) appartient à `Dico(arbre)` si et seulement si la racine de l'arbre contient 1.
Écrire la fonction

```
1  int dico_belongs( node *arbre, const char *word )
2
```

qui retourne 1 si le mot `word` appartient à `Dico(arbre)`, c'est-à-dire tous les caractères avant le caractère `'\0'` qui termine `word` sont les lettres `a` et `b` et le mot correspondant appartient à `Dico(arbre)`.
Dans le cas contraire la fonction renvoie 0.

Vecteurs

Exercice 5 :

On définit

```
1  typedef struct{
2    unsigned nb;
3    int value;
4  } item;
5
```

Écrire la fonction

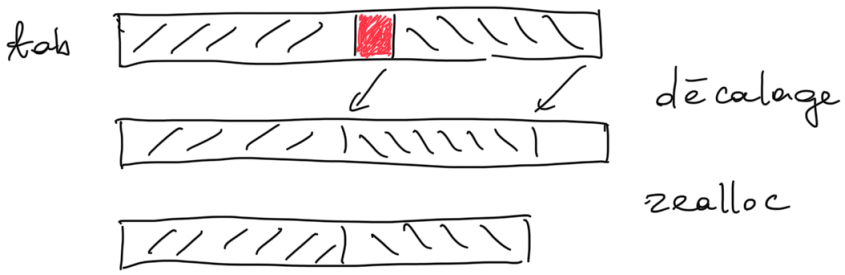
```
1  item *remove( item tab[], int *n, int d)
2
```

qui prend comme paramètre un vecteur `tab` de `item`s. Le paramètre `n` est l'adresse d'un int qui contient le nombre d'éléments de `tab`.
La fonction recherche dans `tab` un item avec la valeur de champ `value` égale à `d`.
Si aucun item de `tab` ne satisfait pas cette condition la fonction retournera `NULL`.
Si `tab` contient un item avec `value == d` alors la fonction décrémente la valeur de champ `nb` de cet item.

- Si la nouvelle valeur de `nb` est strictement positive alors la fonction retournera `tab`.
- Si la nouvelle valeur de `nb` est 0 la fonction supprime le item correspondant en décalant les items qui suivent l'élément à supprimer [1]. Ensuite la fonction raccourcit le vecteur grâce au `realloc`.
  La fonction décrémente la valeur `int` pointée par le paramètre `n` et retourne le pointeur vers le vecteur raccourci.

Le dessin suivant illustre les opérations à effectuer pour supprimer un item (en rouge) :

1. de préférence le décalage doit être effectué avec un simple appel à une fonction du C, sans boucle



2 csv

Le fichier csv est un fichier texte dont le lignes sont composée de champs. Les champs sont séparés par un caractère qu'on appellera délimiteur et qu'on notera `del`. Différents fichiers csv peuvent utiliser différents délimiteurs. On suppose que

- aucun champ ne contient de caractère `del`, `del` sert à séparer le champs mais ne fait pas partie de champs,
- le dernier champ termine par le caractères de la nouvelle ligne `'\n'` et il n'y a pas d'autre occurrences de caractère `'\n'`,
- le délimiteur n'est pas le caractère de nouvelle ligne.

Dans les exemples suivant on assume que `del == ';'`.

Exemples.

`ab␣c\n` – possède un seul champ qui contient `ab␣c`

On suppose que le caractère `'\n'` qui termine toujours le dernier champ ne fait pas partie de ce champ.

`Dupont;␣␣34;␣12.5␣\n` – possède trois champs : le champ 0 : `Dupont`, le champ 1 : `␣␣34` et le champ 2 : `␣12.5␣`.

On suppose que le caractère délimiteur ne fait pas partie de champ.

Notez aussi que nous numéroterons les champs à partir de 0.

`␣␣Damien␣;␣-256\n` – possède deux champs, le champ 0 : `␣␣Damien␣` et le champ 1 : `␣-256`

`␣;;ala;\n` – possède 4 champs, le champ 0 contient un seul caractère d'espace, le champ 1 est vide, c'est-à-dire ne contiennent aucun caractère, le champ 2 contient `ala` et le champ 3 est vide.

Dans cette section il faut écrire quelques fonctions qui permettent de faire des différents traitements de lignes d'un fichiers csv.

Exercice 6 :

Écrire la fonction

```
1  unsigned csv_nb_fields(const char *line, char del)
```

qui retourne le nombre de champs dans la ligne `line` csv. Le paramètre `del` donne le délimiteur de champs.

Ici et dans tous les exercices qui suivent on assume que le paramètre `line` est (un pointeur vers) une chaîne de caractère en C, c'est-à-dire `line` termine avec le caractère nul `'0'`. Mais `line` est aussi une ligne d'un fichier csv, donc

- `line` possède un seul caractère `'\n'` de la nouvelle ligne qui apparaît toujours juste avant le caractère nul `'\0'`,
- `line` contient les occurrences de `del` qui séparent les champs (mais le nombre d'occurrences de `del` peut être zéro si la ligne possède juste un seul champ).

Le paramètre `del` est le délimiteur.

Exercice 7 :

Ecrire la fonction

```
1  char *csv_get_field(const char *line, char del, unsigned n)
```

qui retourne le pointeur vers le premier caractère de `n`-ième champ de la ligne `line`.
Rappelons que on numérote le champ à partir de 0. Donc par exemple si `n == 0` la fonction retourne simplement la valeur `line`.
La fonction retourne NULL si le `n`-ième champ n'existe pas.

Exercice 8 :

Ecrire la fonction

```
1  int csv_field_len(const char *line, char del, unsigned n)
```

qui retourne la longueur (le nombre de caractères) du `n`-ième champs, ou -1 si ce champ n'existe pas.

Exercice 9 :

Ecrire la fonction

```
1     char *csv_extract_field(const char *line, char *del, unsigned n)
2
```

qui retourne le `n`-ième champ de la ligne `line`.
Notez que vous ne devez pas modifier la chaîne de caractères pointée par `line` (`const` devant le paramètre indique que `line` ne sera pas modifié).
La valeur retournée doit être une chaîne de caractères dans le sens du C (terminé par le caractère nul `'\0'`).
Par exemple

`csv_extract_field("␣ala;␣133;␣Damien\n", ';' , 0)`
retournera `"␣ala"`,
`csv_extract_field("␣ala;␣133;␣Damien\n", ';' , 1)`
doit retourner `"␣123"` tandis que
`csv_extract_field("␣ala;␣133;␣Damien\n", ';' , 2)`
retournera `"␣Damien"`.