DS n° 3 Informatique III

Calculatrice et documents non autorisés

Lorsqu'aucune consigne n'est précisée, les réponses peuvent être données en langage C ou en pseudo-code.

Vous pouvez définir autant de fonctions que vous le souhaitez dans chaque exercice. Il n'y a pas nécessairement l'équivalence : une question = une fonction.

Si aucune indication n'est donnée quant à l'existence d'une fonction, cela veut dire que vous devez la recoder. L'utilisation de fonctions de la bilibothèque standard est autorisée uniquement sur consigne explicite dans CHAQUE exercice.

Le barême est donné à titre indicatif mais peut être sujet à changement.

Le barême contiendra des **points négatifs** pour tous les aspects redondants de la matière. Que ce soit syntaxiquement ou fonctionnellement, les éléments de code qui doivent être triviaux à ce stade de l'année n'apporteront pas de points mais peuvent en faire perdre si ils sont mal fait ou inexistants (mauvais prototypes, oublis de déclaration ou d'initialisation, pas de vérification de pointeurs avant utilisation, pas de libération de mémoire allouée dynamiquement, ...).

Un code robuste ET une syntaxe propre du langage C sont requis pour ce devoir.

Exercice 1 (*Restaurateur geek*) *Thème : Listes-Files-Piles* (7.6 pts)

Un restaurateur souhaite utiliser un programme pour gérer les commandes des clients de son retaurant. Comme tous ses clients arrivent au même moment, il se retrouve à devoir préparer l'ensemble des plats au même moment. Il a donc l'habitude de préparer d'abord toutes les entrées, puis tous les plats, puis tous les desserts.

Il souhaite qu'une structure de type liste chainée un peu particulière soit utilisée. En effet il voudrait que dans cette structure, l'insertion dépende du type de plat commandé, et la suppression se fasse simplement dans l'ordre (du debut jusqu'à la fin).

Il existe trois types de plats : ENTREE, PLAT, DESSERT. Pour l'insertion, il voudrait que les entrées soient toujours insérées en début de liste, les desserts toujours en fin de liste, et les plats principaux soient insérés entre les entrées et les desserts. Si deux commandes sont de meme type, leur ordre relatif n'importe pas. On souhaite absolument utiliser une, et une seule, liste dans ce programme.

Chaque commande possède un nom (chaine de caractères allouée dynamiquement) et un type (énoncé plus haut, cela peut être une valeur entière constante).

A tout instant, le programme pourrait obtenir, à temps constant, le nombre de commandes de type ENTREE, ou bien le nombre de commandes de type PLAT, ou bien le nombre de commandes de type DESSERT, ou bien la somme de toutes ces valeurs.

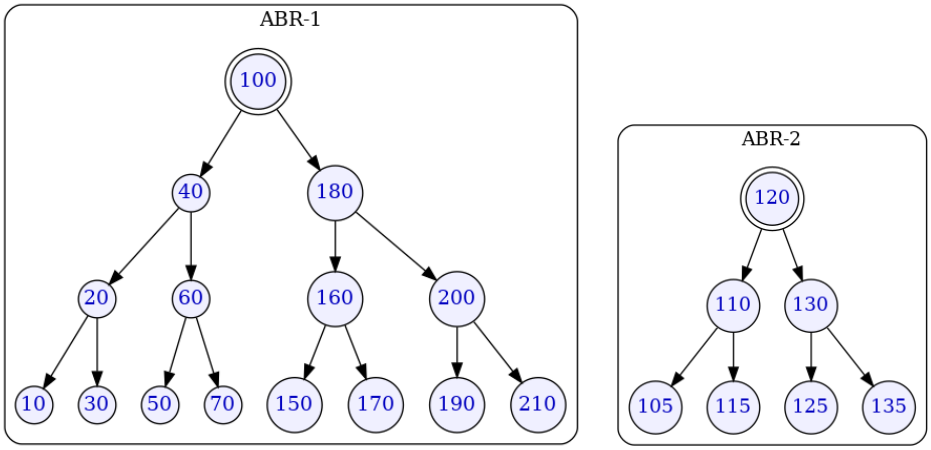
Dans cet exercice, il est autorisé d'utiliser les fonctions du module <string.h>.

Ou de déclarer des prototypes de fonctions qui **manipulent des chaines de caractères** en indiquant ce qu'elles font, et d'utiliser ces fonctions directement dans votre code sans avoir à les coder.

1. Déclarer la ou les structures pour gérer les commandes en respectant le cahier des charges ci-dessus. Créer également la ou les constantes nécessaires.
2. Coder la fonction/procédure `initAllOrders(...)` pour initialiser la structure qui stocke toutes les commandes du restaurant.
3. Coder la fonction/procédure `createOneOrder(...)`pour créer une commande en mémoire. Si un problème est rencontré, le programme doit s'arrêter
4. Coder une fonction/procédure `addOneOrder(...)` pour ajouter une commande d'un client dans la structure qui stocke l'ensemble des commandes du restaurant. Ce bout de programme prendra en entrée la structure qui stocke toutes les commandes, le nom d'une commande, et son type. Il renverra également un booléen pour indiquer si l'ajout s'est bien déroulé ou non. Si un problème est rencontré, la valeur 0 doit être retournée.

Exercice 2 (*Une fusion d'ABR un peu spéciale*) *Thème : ABR* (4.2 pts)

Dans cet exercice nous souhaitons fusionner 2 arbres de type ABR (on suppose que ces arbres sont bien des ABR). Cette fusion est un peu spéciale. Sur les images ci-dessous, on souhaite donc insérer l'ABR-2 en dessous d'un noeud de l'ABR-1 passé en argument.



Si le noeud passé n'existe pas, aucune modification de l'ABR-1 aura lieu.

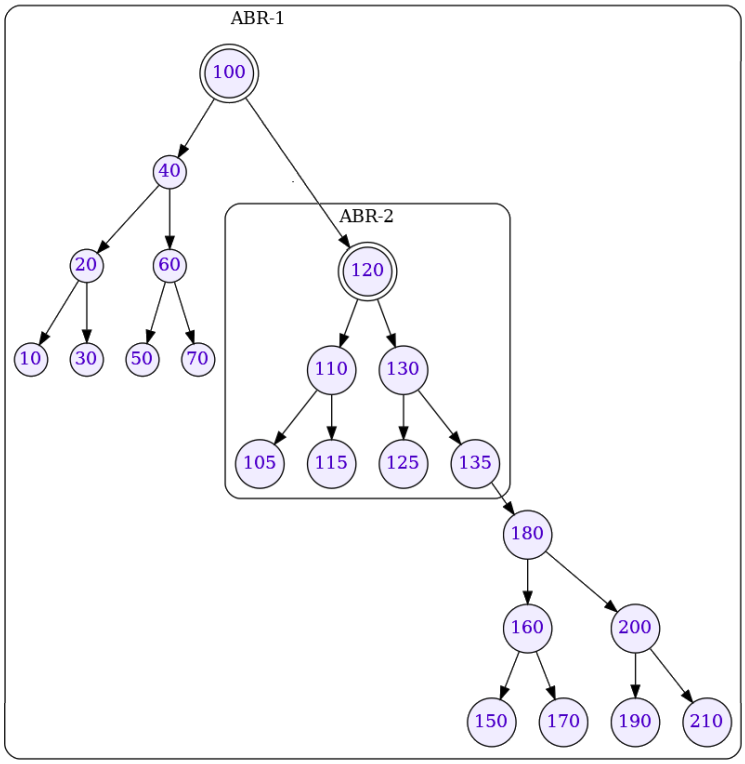
Si le noeud de l'ABR-1 demandé existe, il va falloir insérer l'ABR-2 DIRECTEMENT entre le noeud demandé et son fils (voir détails plus bas). Si le noeud demandé n'existe pas, aucune insertion n'aura lieu.

Il faut donc vérifier que TOUS les noeuds de l'ABR-2 sont cohérents pour faire cette insertion.

Voici quelques étapes sur le fonctionnel demandé :

- A chaque noeud de l'ABR-1 parcouru, il va falloir vérifier que l'ensemble de l'ABR-2 respecte les critères d'insertion ABR. C'est à dire que si l'on souhaite parcourir le fils gauche d'un noeud de l'ABR-1, l'ensemble des valeurs de l'ABR-2 doivent être inférieures strictement. Si ce n'est pas le cas, l'insertion ne pourra pas se faire.
- Si on trouve le noeud demandé (qui doit devenir le noeud parend de l'ABR-2), par exemple ici le noeud 100, il faut déterminer si l'ABR-2 est un sous-arbre gauche ou droit du noeud 100. Ici dans l'exemple, l'ABR-2 peut devenir un fils droit du noeud 100 car toutes ses valeurs sont strictement supérieures.
- Maintenant que l'emplacement est trouvé, pour réaliser l'insertion voulue, il va falloir vérifier que le fils droit actuel du noeud 100 est donc strictement supérieur à la valeur maximale de l'ABR-2. Comme c'est le cas ici, le noeud 120 deviendra le fils droit du noeud 100, et le noeud 180 deviendra le fils droit du noeud 135 (valeur maxi de l'ABR-2)
- Pour résumer, l'insertion de ABR-2 doit se faire comme si c'était un noeud unique donc la valeur serait comprise entre celle donnée en paramètre (noeud cible parent) et celle du fils qui va être décalé dans la topologie.

Voici le résultat de l'insertion de ABR-2 sous le noeud 100 de ABR-1.



On donne la structure de l'ABR :

```
typedef struct node{
    int          value;
    struct node* pLeft;
    struct node* pRight;
} ABR;
```

On suppose qu'il existe une fonction `ABR* getMin(ABR* p);` qui prend en entrée une adresse d'un noeud ABR et qui renvoie l'adresse du noeud contenant la valeur la plus petite.

On suppose qu'il existe une fonction `ABR* getMax(ABR* p);` qui prend en entrée une adresse d'un noeud ABR et qui renvoie l'adresse du noeud contenant la valeur la plus grande.

1. Ecrire une fonction/procédure mergeABR(...) qui prend en paramètre 2 ABR (ABR1 à modifier, et ABR2 à insérer dans ABR1) et prend en paramètre également la valeur d'un noeud de ABR1 qui sera le futur parent de ABR2 (on rappelle que l'on souhaite insérer ABR2 sous ce noeud). Si l'un des 2 arbres est vide, on ne fait rien. Ce bout de programme ne fera rien si il rencontre une erreur. Il se terminera silencieusement. On ne souhaite pas qu'il y aie de doublons une fois la fusion effectuée.

Ce bout de programme ne doit rien faire si une erreur est rencontrée. Il doit également avoir la meilleure complexité possible : stockez vos résultats de calculs intermédiaires pour éviter de les refaire systématiquement. Il est rappelé qu'après la fusion des 2 arbres, l'arbre résultant doit toujours être un ABR !

Exercice 3 (*Equilibrage d'un AVL*) *Thème : AVL* (4.4 pts)

Dans cet exercice, nous souhaitons coder une fonction qui va permettre de vérifier l'équilibrage d'un noeud d'un arbre binaire de type AVL. Nous partons du principe qu'au moment de l'appel de cette fonction tous les noeuds de l'arbre possèdent une information sur le facteur d'équilibre qui est à jour.

On suppose que les fonctions suivantes existent et qu'il n'est pas nécessaire de les recoder :
— `int min2(int a, int b);` → retourne la valeur minimale entre a et b
— `int min3(int a, int b, int c);` → retourne la valeur minimale entre a, b et c
— `int max2(int a, int b);` → retourne la valeur maximale entre a et b
— `int max3(int a, int b, int c);` → retourne la valeur maximale entre a, b et c

1. Créer la structure pour un noeud d'arbre binaire de type AVL contenant comme élément une valeur entière.
2. Créer la fonction/procédure `rotationGauche(...)` qui va effectuer la rotation simple à gauche d'un noeud de l'arbre.
   — Ce bout de programme va devoir modifier les liens entre les noeuds pour garantir la roation
   — Ce bout de programme va mettre à jour les valeurs d'équilibre des noeuds impactés.
   — Pour la mise à jour des valeurs d'équilibre, soit le code est explicite pour détailler tous les cas possibles (if ... else if ... / switch ... case ...) soit une formule compacte est utilisée comme dans le cours mais il faudra rajouter une explication en commentaire pour indiquer d'où provient cette formule. Il est important ici d'avoir une vision des différents cas de mise à jour des facteurs d'équilibre. Une formule compacte donnée sans explication ne suffira pas.
3. Donner le prototype de la fonction/procédure `rotationDroite(...)` pour la rotation simple à droite (et seulement son prototype). Il n'est pas nécessaire de la coder.
4. Créer la fonction/procédure `equilibrageAVL(...)` avec les indications suivantes :
   — Ce bout de programme prend en paramètre l'adresse d'un sous-arbre : c'est l'adresse du noeud qui doit potentiellement être rééquilibré.
   — Ce bout de programme vérifie les valeurs des facteurs d'équilibre et effectue les opérations nécessaires pour rééquilibrer le sous-arbre en fonction de ces valeurs.
   — Ce bout de programme renvoie l'adresse du sous-arbre potentiellement modifié.

Exercice 4 (*Gestion de fichiers*) *Thème : Shell+Unix* (5.2 pts)

1. Ecrire un script shell `displayFiles.sh` récursif qui va parcourir l'ensemble des fichiers d'un dossier passé en argument, et ce de manière récursive (le script doit parcourir toute l'arborescence du dossier passé en argument).

   Le script, au moment du parcours du contenu du dossier, quand il trouve un fichier, doit simplement afficher son nom (juste le nom, pas le chemin complet), et quand il trouve un dossier, il doit afficher son contenu (appel récursif au script) mais NE PAS afficher le nom de ce dossier.

   Pour résumer, ce script affiche les noms de tous les fichiers contenus dans un dossier et ses sous-dossiers.
   Si le script rencontre un problème, il doit s'arrêter et renvoyer un code d'erreur quelconque supérieur à 0.

2. En utilisant le script `displayFiles.sh` de la question précédente, écrire un script `displayDuplicate.sh` qui va détecter les fichiers portant le meme nom.
   Quand un fichier en doublon est détecté, le script doit simplement afficher son nom (une seule fois).
   On n'affichera pas les chemins complets, juste les noms.
   Si le script rencontre un problème, il doit s'arrêter et renvoyer un code d'erreur.

   ```
   [FILES]
     +-archive.tar.gz
     +--[DOCUMENTS]
     |     +-cv.pdf
     |     +-user_guide.pdf
     |     +-taxes.pdf
     +-f500d5zsfre.data
     +--[IMAGES]
     |     +-holidays.png
     |     +-party.jpg
     +--[MISC]
     |     +-archive.tar.gz
     |     +-cv.pdf
     |     +-main.c
     |     +-party.jpg
     |     +-script.sh
     +--[VIDEOS]
           +-wedding.mkv
   ```

   Ici dans l'arborescence présentée, les nom des fichiers `archive.tar.gz`, `cv.pdf` et `party.jpg` doivent apparaitrent si on lance le script en passant comme argument le nom du dossier `FILES`, car ces 3 fichiers apparaissent plusieurs fois dans l'arborescence complète.

3. Creer un script `moveBigFiles.sh` qui va lister tous les fichiers volumineux d'un dossier et les déplacer.
   Ce script va prendre en argument le chemin d'un dossier et une taille maximale de fichier.
   Ce script doit parcourir le dossier passé en argument (uniquement les fichiers du dossier, pas les sous-dossiers) et récupérer les tailles de chaque fichier. Ces tailles seront comparées à la taille maximale passée en argument.
   Tous les fichiers qui dépassent strictement cette valeur devront être déplacés dans un sous-dossier `big_files` du dossier passé en argument, que le script devra créer si il n'existe pas.
   Si le script rencontre un problème, il doit s'arrêter et renvoyer un code d'erreur.

Exemple d'affichage d'une commande 'ls -l' :

```
total 1400
-rwxrwxr-x  1 cytech cytech    404 janv. 21  2022 autogen.sh
drwxrwxr-x  5 cytech cytech   4096 janv. 21  2022 build
-rw-rw-r--  1 cytech cytech 107183 janv. 21  2022 CMakeLists.txt
```

Extrait de la documentation de la commande 'tail' :

```
Usage: uniq [OPTION]... [INPUT [OUTPUT]]
Filter adjacent matching lines from INPUT, writing to OUTPUT.
  -c, --count           prefix lines by the number of occurrences
  -f, --skip-fields=N   avoid comparing the first N fields
```

```
  -i, --ignore-case     ignore differences in case when comparing
  -s, --skip-chars=N    avoid comparing the first N characters
  -u, --unique          only print unique lines
```

Extrait de la documentation de la commande 'tr' :

```
Usage: tr [OPTION]... SET1 [SET2]
Translate, squeeze, and/or delete characters from standard input, writing to standard output.
  -c, -C, --complement    use the complement of SET1
  -d, --delete            delete characters in SET1, do not translate
  -s, --squeeze-repeats   replace each sequence of a repeated character
                            that is listed in the last specified SET,
                            with a single occurrence of that character
  -t, --truncate-set1     first truncate SET1 to length of SET2
```

Extrait de la documentation de la commande 'head' :

```
Usage: head [OPTION]... [FILE]...
Print the first 10 lines of each FILE to standard output.
Mandatory arguments to long options are mandatory for short options too.
  -c, --bytes=[-]NUM       print the first NUM bytes of each file;
                             with the leading '-', print all but the last
                             NUM bytes of each file
  -n, --lines=[-]NUM       print the first NUM lines instead of the first 10;
                             with the leading '-', print all but the last
                             NUM lines of each file
```

Extrait de la documentation de la commande 'tail' :

```
Usage: tail [OPTION]... [FILE]...
Print the last 10 lines of each FILE to standard output.
Mandatory arguments to long options are mandatory for short options too.
  -c, --bytes=[+]NUM       output the last NUM bytes; or use -c +NUM to
                             output starting with byte NUM of each file
  -n, --lines=[+]NUM       output the last NUM lines, instead of the last 10;
                             or use -n +NUM to output starting with line NUM
```

Extrait de la documentation de la commande 'cut' :

```
Usage: cut OPTION... [FILE]...
Print selected parts of lines from each FILE to standard output.
Mandatory arguments to long options are mandatory for short options too.
  -d, --delimiter=DELIM   use DELIM instead of TAB for field delimiter
  -f, --fields=LIST       select only these fields;  also print any line
                            that contains no delimiter character, unless
                            the -s option is specified
      --complement        complement the set of selected bytes, characters
                            or fields
```

Extrait de la documentation de la commande 'sort' :

```
Usage: sort [OPTION]... [FILE]...  or: sort [OPTION]... --files0-from=F
Write sorted concatenation of all FILE(s) to standard output.
Ordering options:
  -b, --ignore-leading-blanks  ignore leading blanks
  -d, --dictionary-order      consider only blanks and alphanumeric characters
  -f, --ignore-case           fold lower case to upper case characters
  -n, --numeric-sort          compare according to string numerical value
  -r, --reverse               reverse the result of comparisons
  -k, --key=KEYDEF          sort via a key; KEYDEF gives location and type
  -t, --field-separator=SEP  use SEP instead of non-blank to blank transition
  -u, --unique
```

Extrait de la documentation de la commande 'du' :

```
Usage: du [OPTION]... [FILE]...
   or:  du [OPTION]... --files0-from=F
Summarize disk usage of the set of FILEs, recursively for directories.
  -b, --bytes           equivalent to '--apparent-size --block-size=1'
  -c, --total           produce a grand total
  -h, --human-readable  print sizes in human readable format (e.g., 1K 234M 2G)
```