Nom : _____

Prénom : _____

Numéro étudiant : | | | | | | | | |

Les listes chainées

12 novembre 2020

A lire absolument :

1. L'objectif n'est pas d'apprendre la correction par cœur, mais de comprendre les mécanismes mis en œuvre. Cela vous permettra de vous adapter face à un problème nouveau.
2. En particulier, vous devez être capable de refaire l'intégralité du sujet, seul, sans aucune aide ni support.
3. Votre travail sera corrigé automatiquement par l'outil de correction automatique CAT. Cela implique que vous devez respecter scrupuleusement les consignes de chaque exercice. Faites très attention aux messages qu'il vous est demandé d'afficher. Un espace en trop, un saut de ligne en moins et l'exercice risque d'échouer.
4. L'enseignant voit votre activité sur le site, ainsi que l'historique de vos dépôts. Pensez à déposer votre travail régulièrement afin qu'il puisse vous apporter des conseils personnalisés.
5. Si l'enseignant vous demande de rendre votre travail sur papier, vous devez répondre directement sur le sujet en respectant absolument la zone prévue à cet effet. Tout ce qui se trouve en dehors de la zone sera ignoré.
6. Si le sujet contient un QCM, vous devez colorier les cases avec un stylo bleu ou noir. Les autres couleurs seront ignorées.
7. Chaque feuille est identifiée de manière unique. Vous pouvez donc rendre votre sujet avec les feuilles mélangées, mais il est préférable de les trier car cela vous permet de vérifier que vous n'en avez pas oublié une.
8. Si vous faites face à un problème, un bug, une erreur ou que vous souhaitez participer à l'amélioration de la plateforme, envoyez un mail à l'adresse suivante : support-cat@liste.parisnanterre.fr

Ne rien écrire dans cette zone

Définition d'une liste chainée

Définissez un nouveau type de donnée « struct liste » permettant d'enregistrer un entier, nommé « valeur » ainsi qu'un pointeur vers une structure du même type nommé « next ». Renommez ensuite ce nouveau type en « liste ».

Création d'un maillon

Écrire une fonction qui attend un entier en argument, puis alloue dynamiquement un élément de type liste, affecte la valeur passée en argument au champs *valeur* et initialise le champs *next* à NULL. La fonction doit retourner l'adresse de cet élément.

```
1 liste * creation_maillon(int n);
```

Est-ce que la liste est vide ?

Écrire une fonction qui accepte en argument l'adresse d'une liste. Votre fonction doit retourner 1 si la liste est vide, 0 sinon.

```
1 int est_vide(liste * l);
```

Affichage itératif des éléments d'une liste chainée

Écrire une fonction itérative qui affiche la valeur du champs *valeur* pour chaque élément d'une liste chainée constituée d'éléments de type liste. L'affichage, sans espace ni saut de ligne doit suivre le format suivant : $val_1 -> val_2 -> val_3 -> \ldots -> val_n -> NULL$, avec val_i la valeur du champs *valeur* du i_eme élément de la liste chainée.

```
1 void affichage_liste1(liste * l);
```

Affichage récursif des éléments d'une liste chainée

Écrire une fonction récursive qui affiche la valeur du champs *valeur* pour chaque élément d'une liste chainée constituée d'éléments de type liste. L'affichage, sans espace ni saut de ligne doit suivre le format suivant : $val_1 -> val_2 -> val_3 -> \ldots -> val_n -> NULL$, avec val_i la valeur du champs *valeur* du i_eme élément de la liste chainée.

```
1 void affichage_liste_rec(liste * l);
```

Affichage dans l'ordre inverse des éléments d'une liste chainée

Écrire une fonction récursive qui affiche la valeur du champs *valeur* pour chaque élément d'une liste chainée constituée d'éléments de type liste. L'affichage, sans espace ni saut de ligne doit suivre le format suivant : $NULL < -val_n < -val_{n-1} < -val_{n-2} < - \ldots < -val_1$, avec val_i la valeur du champs *valeur* du i_eme élément de la liste chainée.

```
1 void affichage_liste_rec_a_l_envers(liste * l);
```

Ajout en tête d'une liste chainée

Écrire une fonction qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste, ainsi qu'un pointeur vers un élément de type liste e et renvoie une liste chainée ayant comme premier élément e et dont la suite est constituée par la liste l.

```
1 liste * ajoute_tete(liste * l, liste * e);
```

Suppression en tête de liste

Écrire une fonction qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste et supprime le premier maillon de cette chaine (n'oubliez pas la libération de la mémoire de ce maillon) puis retourne un pointeur vers le premier élément du reste de la liste. Si le reste de la liste est vide, retourner $NULL$.

```
1 liste * suppression_tete(liste * l);
```

Ajout en fin de liste, version itérative

Écrire une fonction itérative qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste, ainsi qu'un pointeur vers un élément de type liste (e). Cette fonction doit ajouter l'élément e à la fin de la liste et retourner l'adresse du premier élément de cette nouvelle liste.

```
1 liste * ajoute_queue(liste * l, liste * e);
```

Ne rien écrire dans cette zone

Ajout en fin de liste, version récursive

Écrire une fonction récursive qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste, ainsi qu'un pointeur vers un élément de type liste (e). Cette fonction doit ajouter l'élément e à la fin de la liste et retourner l'adresse du premier élément de cette nouvelle liste.

```
1 liste * ajoute_queue_rec(liste * l, liste * e);
```

Ne rien écrire dans cette zone

Suppression en fin de liste, version itérative

Écrire une fonction itérative qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste et supprime le dernier maillon de cette chaine (n'oubliez pas la libération de la mémoire de ce maillon) et retourne un pointeur vers le premier élément de la liste obtenue. Si la liste obtenue est vide, retourner NULL.

```
1 liste * suppression_queue(liste * l);
```

Ne rien écrire dans cette zone

Suppression en fin de liste, version récursive

Écrire une fonction récursive qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste et supprime le dernier maillon de cette chaine (n'oubliez pas la libération de la mémoire de ce maillon) et retourne un pointeur vers le premier élément de la liste obtenue. Si la liste obtenue est vide, retourner NULL.

```
1 liste * suppression_queue_rec(liste * l);
```

Ne rien écrire dans cette zone

Recherche itérative d'une valeur dans une liste chainée

Écrire une fonction itérative qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste ainsi qu'un entier e et qui retourne l'adresse du premier maillon contenant la valeur e. Si cette valeur n'est pas présente dans la liste, retourner NULL.

```
1 liste * recherche(liste * l, int e);
```

Recherche récursive d'une valeur dans une liste

Écrire une fonction récursive qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste ainsi qu'un entier e et qui retourne l'adresse du premier maillon contenant la valeur e. Si cette valeur n'est pas présente dans la liste, retourner NULL.

```
1 liste * recherche_rec(liste * l, int e);
```

Suppresion de la première occurence d'une valeur dans une liste chainée, version récursive

Écrire une fonction récursive qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste ainsi qu'un entier e. Cette fonction doit supprimer le premier élément de la liste dont la valeur du champs *valeur* vaut e, et retourne un pointeur vers le premier élément de la liste ainsi modifiée.

```
1 liste * efface_premiere_occ_rec(liste * l, int e);
```

Libérée, délivrée, version itérative

Écrire une fonction itérative qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste et libère la mémoire de tous les éléments qui la composent. Attention, lorsque vous libérez la mémoire d'un maillon, vous n'avez plus le droit d'accéder à ses différents champs. Vous devez donc sauvegarder l'adresse du maillon suivant.

```
1 void liberation(liste * l);
```

Ne rien écrire dans cette zone

Libérée, délivrée, version récursive

Écrire une fonction itérative qui accepte en argument un pointeur vers une liste chainée constituée d'éléments de type liste et libère la mémoire de tous les éléments qui la composent. Attention, lorsque vous libérez la mémoire d'un maillon, vous n'avez plus le droit d'accéder à ses différents champs. Vous devez donc faire attention à l'ordre dans lequel vous réalisez l'appel récursif et la libération du maillon.

```
1 void liberation_rec(liste * l);
```

Ne rien écrire dans cette zone