Pré-ING2
Semestre 1 - 2022/2023

DS n° 1 Informatique III

Calculatrice et documents non autorisés

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

Exercice 1 (*Opérations sur les files/piles*) (1,5 pts)

Nous disposons de 2 structures ordonnées : P1, une `pile` d'éléments entiers, et F1, une `file` d'éléments entiers. Ces 2 structures sont vides au départ de l'exercice.

Le but de cet exercice est d'afficher l'état des 2 structures `après` chacune des étapes `numérotées` décrites ci-dessous :

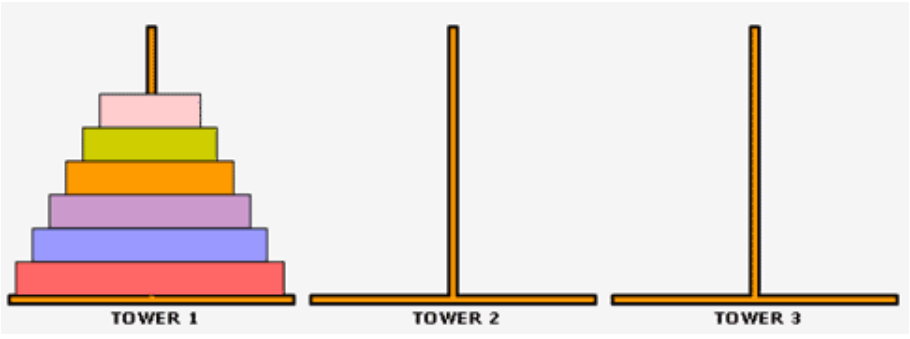
1. empiler dans P1 les valeurs suivantes dans l'ordre : 5, 14, 42, 51, 67, 23
   enfiler dans F1 les valeurs suivantes dans l'ordre : 1, 17, 28, 59, 60, 57
2. dépiler P1, et enfiler la valeur récupérée dans F1
3. défiler F1, et empiler la valeur récupérée dans P1
4. dépiler P1, 2 fois
5. défiler F1, 2 fois
6. dépiler P1, et enfiler la valeur récupérée dans F1
7. défiler F1, et empiler la valeur récupérée dans P1
8. enfiler dans F1 la valeur 99
9. défiler F1 et empiler la valeur récupérée dans P1
10. dépiler P1 et enfiler la valeur récupérée dans F1

Exercice 2 (*Liste doublement chaînée*) (6,0 pts)

On rappelle qu'un élément d'une liste chaînée possède un lien vers l'élément suivant et vers l'élément précédent

1. Ecrire une structure `ChainonD` permettant de gérer une liste doublement chaînée qui va contenir des caractères.
2. Ecrire une fonction `creationChainonD(char nb)` permettant de créer un nouveau `ChainonD` qui contient la valeur `nb` et qui retourne son adresse.
3. Ecrire une fonction `insertPos(ChainonD* tete, int pos, char c)` qui insèrera un nouveau `ChainonD` dans la liste pointée par `tete` à la position `pos` (si pos=0, on mettra le nouveau chaînon au début de la liste, si pos=1, on le placera en deuxième position etc.). Si la position indiquée est plus grande que le nombre de ChainonD, on placera le nouvel element tout à la fin de la liste.
4. Ecrire une procédure `afficheMaj(ChainonD *tete` qui parcourt la liste chaînée et affiche uniquement les majuscules stockées dans la liste.

Exercice 3 (*Tours de Hanoï*) (10,0 pts)



Les tours de Hanoï est un jeu de réflexion consistant à déplacer des disques de diamètres différents d'une tour de «départ» à une tour d'«arrivée» en passant par une ou plusieurs tours «intermédiaires», et ceci en un minimum de coups, tout en respectant les règles suivantes :

— On ne peut déplacer plus d'un disque à la fois
— On ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide

On suppose que cette dernière règle est également respectée dans la configuration de départ illustrée par la figure ci-dessous :

Dans cet exercice, nous représenterons les disques par des entiers dont la valeur représente le diamètre.

1. Quelle méthode de stockage de données informatique permet de représenter au mieux le fonctionnement d'une tour de Hanoi (on considérera que le nombre de disques peut varier d'une partie à l'autre et n'est pas limité).
2. Définir la structure `Tour` qui devra, en plus des champs habituels, inclure un champ `taille` représentant le nombre de disques présents dans une `Tour`.
3. Ecrire une fonction `enleverDisque(...)` permettant de retirer un disque d'une `Tour` dont l'adresse sera passée en paramètre. Cette fonction doit vérifier le bon fonctionnement de la `Tour` et des règles du jeu. Le reste du programme doit pouvoir avoir accès à la valeur du diamètre du disque retiré.
4. Ecrire une fonction `ajouterDisque(...)` permettant d'ajouter à une `Tour` (dont l'adresse sera passée en paramètre) un disque de diamètre `d` (également passé en paramètre). Cette fonction doit vérifier le bon fonctionnement de la Tour et des règles du jeu.
5. Ecrire une procédure permettant d'afficher le contenu d'une `Tour` (les valeurs des diamètres de ses disques) verticalement avec le disque le plus petit au-dessus, comme on pourrait le visualiser sur une vraie tour.
   Exemple possible :

```
 2
 3
 8
13
87
```

6. Ecriture du programme principal :
   — Déclarer 3 `Tour` a, b, c qui seront initialement vides.
   — Demander à l'utilisateur le nombre de disques du jeu, sachant que ce nombre doit être compris entre 3 et 10.
   — Les disques de jeu, représentés par des entiers, auront des valeurs entre 1 et le nombre de disque choisis. Remplir la `Tour` a avec tous les disques du jeu en respectant les règles.
   — Le jeu enchaine donc les cycles de jeu en respectant les règles. Un cycle de jeu se déroule comme suit :
      — On demande à l'utilisateur sur quelle `Tour` il souhaite retirer un disque puis sur quel `Tour` il souhaite le replacer.
      — Si cela est possible le déplacement du disque est effectué
      — On affiche le contenu des trois `Tour` de gauche à droite (a puis b puis c)
   — Le jeu s'arrête lorsque tous les disques sont sur la `Tour` c (celle qui est le plus à droite)
   — Ecrire le programme permettant donc de jouer au jeu et de s'arrêter en cas de victoire.
7. **[BONUS] (+2,0 pts)** Réécrire le programme principal pour que le jeu ne se fasse plus sur 3 `Tour` mais sur un nombre de `Tour` dynamique entre 3 et 20 qui sera choisi par l'utilisateur. On utilisera alors un tableau de `Tour` pour gérer le jeu. Le nombre de disques quant à lui pourra être compris entre 3 et 500.

Exercice 4 (*Séparation d'une liste en 2*) **(2,5 pts)**

On souhaite écrire une fonction `coupeEn2(...)` **en langage C** qui prend en entrée une liste chaînée, et qui va "couper" cette liste en 2 sous-listes de tailles égales.

— Chaque élément d'une liste sera typé par le type `Chainon` à définir comme suit :
   — une valeur entière comme valeur de l'élément,
   — un pointeur sur le chainon suivant.
— Cette fonction va utiliser la valeur de retour pour renvoyer un booléen afin d'indiquer si l'opération a réussi ou non :
   — si la taille de la liste d'entrée est paire (ou vide), alors la fonction pourra effectuer la séparation, et renverra '1' pour indiquer que la séparation a fonctionné (elle pourra éventuellement renvoyer '0' si une erreur survient au cours de la séparation),
   — sinon elle renverra '0' pour indiquer que la taille de la liste est impaire et que la séparation ne peut pas être effectuée.
— Cette fonction doit donc "couper" la liste d'entrée en 2 sous-listes de tailles égales : cela signifie que la fonction appelante doit pouvoir récupérer les 2 adresses des 2 sous-listes. La valeur de retour est déjà utilisée, il convient donc d'utiliser les paramètres pour retourner ces 2 adresses.
— On part du principe qu'il existe une fonction `int taille(Chainon* p)` qui prend une liste chaînée en entrée et qui retourne le nombre d'éléments de cette liste. Il n'est pas nécessaire de recoder cette fonction.