UNIVERSITE CHEIKH ANTA DIOP DE DAKAR

ECOLE SUPERIEURE POLYTECHNIQUE

DEPARTEMENT INFORMATIQUE

DIC 1 && Licence GL Langage C

TD & TP N° 3 [1] [2] [3]

PARTIE I

Exercice 1

Soit la déclaration d'une liste d'entiers (voir annexe)

1- On vous demande d'écrire les procédures (pas de type de retour) permettant de :
   - Créer une liste vide ;
   - Insérer en tête de la liste ;
   - Insérer en fin de liste ;
   - Supprimer en fin de liste ;
   - Afficher une liste à l'envers ;
2- Il vous est demandé d'écrire deux versions (fonction et procédure) pour ces routines suivantes :
   - Supprimer en début de liste ;
   - Rechercher un élément donné dans une liste ;
   - Supprimer un élément donné dans une liste ;
   - Insérer un élément dans une liste triée ;

[1] Prenez le fichier en annexe pour vous aider un peu

[2] Si vous vous attendez à une correction de notre part vous risquez d'être déçus. ☹

Exercice 2

Considérant la même déclaration de liste :
On vous demande d'écrire une fonction permettant de

- Faire l'union de deux listes ;
- Faire l'union de deux listes sans doublons ;
- Faire l'intersection de deux listes ;
- Faire la fusion de deux listes triées.

Exercice 3

Faites la déclaration d'une liste doublement chainée, c'est-à-dire une liste telle qu'un élément pointe à la fois sur son suivant et son précédent.
Reprendre l'exercice précédent en utilisant ce formalisme.

Exercice 4

Faites la déclaration d'une liste circulaire, c'est-à-dire une liste qui n'a pas de fin. Le dernier élément pointe sur le premier.
Reprendre l'exercice 1 en utilisant ce formalisme.

Exercice 5

1 Rechercher l'élément maximal d'une liste **

Écrire deux algorithmes, l'un itératif, l'autre récursif, qui permettent de déterminer la valeur maximale d'une liste chaînée d'entiers positifs.

2 Concaténer deux listes *

On considère deux listes chaînées L_1 et L_2 dont les éléments sont des entiers. Écrire un algorithme qui rattache la liste L_2 à la suite de la liste L_1. Tous les cas particuliers doivent être pris en compte.

3 Extraire deux listes à partir d'une liste ***

Soit une liste de nombres relatifs. Écrire un algorithme permettant de séparer cette liste en deux listes : la première ne comportant que des entiers positifs ou nuls, et la seconde ne comportant que des nombres négatifs.

4 Permuter deux places d'une liste ***

[3] Bon courage ☺

Écrire un algorithme sur une liste simplement chaînée qui échange les positions des noeuds données par deux pointeurs t et v.

5 Supprimer des éléments **

Soit L une liste chaînée. Écrire trois algorithmes tels que :

• dans le premier on supprime toutes les occurrences d'un élément donné x ;
• dans le deuxième, on ne laisse que les *k* premières occurrences de cet élément et on supprime les suivantes ;
• dans le troisième, pour chaque élément de la liste, on ne laisse que sa première occurrence.

6 Inverser une liste **

Écrire un algorithme qui inverse une liste chaînée sans recopier ses éléments. Réaliser une version itérative et une autre récursive.

7 Construire une liste à partir d'une source de données *

Concevoir un premier algorithme qui construit une liste chaînée linéaire à partir d'un tableau, et un second qui duplique une liste chaînée existante.

8 Refermer une liste sur elle-même *

Concevoir un algorithme qui transforme une liste simplement chaînée linéaire en liste simplement chaînée circulaire.

9 Effectuer et retourner deux calculs sur une liste **

Concevoir un algorithme qui calcule et retourne respectivement le produit des éléments positifs et celui des éléments négatifs d'une liste simplement chaînée d'entiers.

10 Couper une liste en deux **

Concevoir un algorithme qui sépare une liste simplement chaînée d'entiers en deux listes :

• A – à partir d'un pointeur sur le maillon qui devient tête de la seconde liste de sortie ;
• B – juste devant le premier maillon contenant une valeur donnée *x* ;
• C – à partir de la *k*ième position (c'est-à-dire *k* maillons dans la première liste de sortie et le reste
dans la seconde) ;
• D – juste devant le maillon contenant la valeur minimale de la liste ;
• E – telles que les deux listes de sortie aient même longueur *n*, ou que la première soit de longueur
n+1 et la seconde de longueur *n*.

11 Supprimer un sous-ensemble d'une liste *

Concevoir un algorithme qui supprime :

• A – un maillon sur deux, en commençant par la tête de liste,
• B – les maillons contenant des éléments impairs,
• C – les maillons contenant des éléments pairs,
• D – les maillons contenant des éléments supérieurs à un seuil donné,
• E – les maillons contenant des éléments inférieurs à un seuil donné,
d'une liste simplement chaînée (d'entiers).

12 Saisir, enregistrer puis évaluer un polynôme***

Enregistrer les coefficients et les exposants d'un polynôme dans une liste chaînée. Évaluer ce
polynôme pour une valeur donnée de *x*.
Utiliser au moins deux modules dans ce programme :

• l'un pour construire la liste sur la base des coefficients et des exposants, qui sont saisis au clavier ;
• et le second afin d'évaluer le polynôme pour la valeur *x*, également saisie au clavier

PARTIE II

Les piles et les files sont des ensembles dynamiques pour lesquels l'élément à supprimer *via* l'opération SUPPRIMER est défini par la nature intrinsèque de l'ensemble. Dans une *pile*, l'élément supprimé est le dernier inséré : la pile met en œuvre le principe *dernier entré, premier sorti*, ou *LIFO* (Last-In, First-Out). De même, dans une *file*, l'élément supprimé est toujours le plus ancien ; la file met en œuvre le principe *premier entré, premier sorti*, ou *FIFO* (First-In, First-Out). Il existe plusieurs manières efficaces d'implémenter des piles et des files dans un ordinateur. Dans cette section, nous allons montrer comment les implémenter à l'aide d'un tableau simple.

1) Piles

L'opération INSÉRER dans une pile est souvent appelée EMPILER et l'opération SUPPRIMER, qui ne prend pas d'élément pour argument, est souvent appelée DÉPILER. Ces noms font allusion aux piles rencontrées dans la vie de tous les jours, comme les piles d'assiettes

automatiques en usage dans les cafétérias. L'ordre dans lequel les assiettes sont dépilées est l'inverse de celui dans lequel elles ont été empilées, puisque seule l'assiette supérieure est accessible.

2) Files

On appelle ENFILER l'opération INSÉRER sur une file et on appelle DÉFILER l'opération SUPPRIMER ; à l'instar de l'opération de pile DÉPILER, DÉFILER ne prend pas d'argument. La propriété FIFO d'une file la fait agir comme une file à un guichet d'inscription. La file comporte une ***tête*** et une ***queue***. Lorsqu'un élément est enfilé, il prend place à la queue de la file, comme l'étudiant nouvellement arrivé prend sa place à la fin de la file d'inscription. L'élément défilé est toujours le premier en tête de la file, de même que l'étudiant qui est servi au guichet est celui qui a attendu le plus longtemps dans la file. (Heureusement, nous n'avons pas ici à prendre en compte les éléments qui resquillent).

Exercice 1

Implémenter ces différentes fonctions en utilisant un tableau puis une liste :

- pileVide(Pile p) : teste si une pile est vide ou pas
- empiler (p, x) : ajoute x à la pile
- dépiler(p) : enlève un élément de la pile
- fileVide(File f) : teste si une file est vide ou pas ;
- enfiler(f, x) : ajoute x à la file ;
- défiler (f) ;

Exercice 2

Alors qu'une pile n'autorise l'insertion et la suppression des éléments qu'à une seule extrémité et qu'une file autorise l'insertion à une extrémité et la suppression à l'autre extrémité, une ***file à double entrée*** autorise l'insertion et la suppression à chaque bout. Écrire quatre procédures pour insérer et supprimer des éléments de chaque côté d'une file à double entrée construite à partir d'un tableau.

Exercice 3

Expliquer comment implémenter deux piles dans un seul tableau de telle manière qu'aucune pile ne déborde.
Montrer comment implémenter une file à l'aide de deux piles
Montrer comment implémenter une pile à l'aide de deux files

Exercice 4

Concevoir deux algorithmes, qui créent respectivement :
• la file inverse d'une file ;
• la pile inverse d'une pile.
Ces deux algorithmes doivent restituer leur entrée inchangée

Exercice 5

Écrire un algorithme permettant de lire deux entiers positifs *n* et *k*. Construire une liste circulaire dans laquelle seront enregistrés les nombres *1, 2, 3, ..., n*, dans cet ordre. En commençant à partir du noeud contenant *1*, supprimer successivement tous les *kièmes* noeuds de la liste, en effectuant un parcours circulaire dans celle-ci et jusqu'à ce que tous les noeuds aient été supprimés. Dès qu'un
noeud est supprimé, le suivant dans la boucle est considéré comme la nouvelle tête de liste et ainsi de suite. Si $n = 8$ et $k = 3$, par exemple, la suppression des noeuds contenant les huit entiers se fait dans cet ordre :
3, 6, 1, 5, 2, 8, 4, 7
(Ce procédé peut être illustré par un groupe composé à l'origine de *n* personnes formant un cercle, que l'on élimine successivement en désignant le *kième* de ceux restant dans le cercle que l'on continue de parcourir).

Exercice 6

Soit une liste simplement chaînée de type Heqat, c'est-à-dire obtenue par l'opération concat(&l, sublist(&l, n)), concevoir un algorithme pour en mesurer la longueur en nombre de noeuds, qui retourne d'une part la longueur de la partie amont linéaire, et d'autre part la longueur

(périmètre) de la partie circulaire en aval.
Vous devrez respecter les contraintes suivantes :
• pas de modification de la liste pendant le traitement (lecture seule) ;
• pas de copie de l'information des noeuds traversés.