TD 02 : Piles et Files (1)

Consignes générales : N'oubliez pas pour ce TD comme pour les suivants de vous créer un répertoire consacré au TD et d'enregistrer vos codes dedans.

On rappelle que les commandes à taper dans le terminal pour compiler puis éxécuter votre programme C :
— Pour compiler : *gcc -o nom_executable nom_programme.c*
— Pour exécuter : *./nom_executable*

Exercice 1 (*Application du cours*)

1. Soit la **pile** suivante :
   [2, 5, 9, 10, 1, -3]
   avec -3 le haut de la pile. Indiquez l'état de la pile une fois les opérations suivantes effectuées
   — Dépiler
   — Dépiler
   — Empiler 12
   — Dépiler
   — Empiler 8
   — Empiler 7
   (a) Dans cet ordre
   (b) Dans l'ordre inverse
2. Mêmes questions si la liste présentée est une **file** avec l'équivalence empiler/enfiler, dépiler/défiler.
3. Comment pourrait-on simuler une file à l'aide de piles ?

Exercice 2 (*Gestion de piles*)

1. Déclarer la structure `PileDyn` permettant de gérer une pile dynamique contenant des entiers.
2. Ecrire une fonction `empiler(int nb, PileDyn* ppile)` permettant d'empiler `nb` sur la pile dynamique pointée par `ppile`.
3. Déclarer une nouvelle pile `p1` et lui empiler les nombres entiers de 1 à 20 dans l'ordre croissant. Que vaudra le sommet de la pile ?
4. Ecrire une procédure `affichePile(PileDyn * ppile)` permettant d'afficher le contenu d'une pile de manière `récursive`. Afficher `p1`.
5. Ecrire une fonction `depile(PileDyn * ppile, int *pnmb)` permettant de dépiler une pile d'entiers passée en paramètre et de modifier la valeur de la variable pointée par *pnmb* avec la valeur stocké dans l'element supprimé.
6. Déclarer deux autres piles `pilePair` et `pileImpair` qui seront remplies à partir de `p1` de la manière suivante :
   — p1 est dépilée
   — Si l'élément dépilé est pair il est empilé dans `pilePair`
   — Si l'élément dépilé est impair il est empilé dans `pileImpair`
   A quoi vont ressembler ces deux piles ? Vérifier grâce à `affichePile(PileDyn * ppile)`.
7. Réecrire le programme en utilisant une pile **statique**.

Exercice 3 (*Simulations de clients d'un supermarché*)

Nous allons simuler le passage de clients d'un supermarché en caisse de paiement. Chaque client sera simulé par un entier contenant le nombre d'articles de son caddy. Le but de cet exercice sera d'afficher l'ensemble des caddys des clients en train de patienter à une caisse.

1. Quelle structure (tableau, liste chaînée, pile, file) semble la plus appropriée pour ce type d'exercice ?
2. Déclarer ce type de structure avec comme type d'élément un entier.
3. Créer une fonction qui va créer une instance d'un client avec une valeur entière aléatoire entre 1 et 50 (simulant le nombre d'articles dans son caddy).
4. Créer une fonction qui va afficher tous les clients dans l'ordre d'arrivée en caisse.
5. Créer un programme qui va simuler l'arrivée de clients à la caisse :
   — Ajouter 3 clients en caisse

— Dans une boucle infinie, au début de chaque tour, on enlève le client le plus proche de la caisse (le prochain à payer ses achats), si il existe
— Ensuite il y a 33% de chances d'ajouter de nouveaux clients en caisse
— Si on ajoute des clients, suite au point précédent, on en ajoute entre 1 et 3 (de manière aléatoire)
— A la fin de la boucle on affiche le client qui a payé ses achats et on affiche en dessous le reste des clients (dans l'ordre d'arrivée à la caisse)
— Si il n'y a plus de clients en caisse, on arrête le programme

6. Proposer un algorithme pour modifier le programme précédent de la façon suivante :
   — En commençant par les clients les plus proches de la caisse, déterminer ceux qui voudraient changer de caisse si la somme des articles des autres clients situés devant dépasse un certain seuil
   — Est ce que la structure choisie est appropriée pour ce type de calcul ? Justifier la réponse.

Exercice 4 (*Vérification du parenthésage*)

La plupart des traitements de texte ou de calculs sont capables d'analyser la syntaxe et d'indiquer un problème de parenthésage.

Pour vérifier qu'un texte/une expression contient des parenthèses correctes, il ne suffit pas que le nombre de parenthèses ouvrantes soit le même que le nombre de parenthèses fermantes : l'ordre dans lequel on rencontre les parenthèses fermantes doit correspondre à l'ordre dans lequel on a rencontré les parenthèses ouvrantes.

Exemple : *"Je suis Luffy)le futur roi des pirates(!"* a un mauvais parenthésage bien qu'il y ait une parenthèse ouvrante et fermante.

1. Quelle structure (tableau, liste chaînée, pile, file) est la plus adaptée pour vérifier le parenthésage d'une phrase ?
2. Déclarer ce type de structure qui contiendra des caractères.
3. Ecrire un programme permettant de vérifier le bon parenthèsage d'une chaîne de caractères. Vérifier votre algorithme en saisissant une formule mathématique.
4. Faire de même pour des phrases pouvant contenir deux types de symboles : les parenthèses "()" et les crochets "[]". Les règles sont les suivantes :
   https ://fr.overleaf.com/project/62fbe2c4b75d7e9e42e3c150comme précédement, les symboles fermants doivent correspondre à l'ordre dans lequel on a rencontré les symboles ouvrants. le type (parenthèse ou crochet) d'un symbole fermant doit toujours correspondre au type du dernier symbole ouvrant rencontré (et non encore fermé).

   Exemple :
   "a(b[c()e]f)g" est bien parenthésé.
   "a(b[c)d]e" ne l'est pas.