SRI 1ère année - Pile statique

Première partie : pile statique d'entiers

Il s'agit d'implanter la structure de données PILE (LIFO - last in, first out - le dernier entré est le premier sorti).
Une pile statique est représentée par un tableau de taille fixée d'entiers (MAX déclaré comme constante). L'entier en tête de pile (c'est-à-dire le dernier entier inséré dans la pile) est repéré par son indice (tête) dans le tableau. Insérer un entier consiste à ajouter cet entier et à mettre à jour la tête de pile, retirer un entier consiste à supprimer l'entier en tête de pile puis à mettre à jour la tête.

1. Créer un répertoire `TP_Pile_Statique_P1`, s'y placer et créer un fichier `pile_statique_P1.c`.
2. Définir le type *PILE* de façon à avoir le tableau et l'indice de la tête de pile dans la même structure de données.
3. Définir la fonction *init_PILE* qui initialise une pile en respectant le prototype : *PILE init_PILE()*.
4. Définir une fonction *affiche_PILE* qui permet d'afficher tous les entiers d'une pile donnée en paramètre en respectant le prototype : *void affiche_PILE(PILE)*.
5. Définir une fonction *PILE_estVide* qui permet de tester si une pile donnée en paramètre est vide en respectant le prototype : *int PILE_estVide(PILE)*.
6. Définir une fonction *PILE_estPleine* qui permet de tester si une pile donnée en paramètre est pleine en respectant le prototype : *int PILE_estPleine(PILE)*.
7. Définir une fonction *emPILE* qui permet d'empiler un entier (donné en paramètre) dans une pile (donnée en paramètre) en respectant le prototype : *PILE emPILE(PILE, int)*.
8. Définir une fonction *dePILE* qui permet de depiler une pile (donnée en paramètre), cette fonction doit aussi renvoyer l'entier qui était en tête de pile en respectant le prototype : *PILE dePILE(PILE, int *)*.
9. Définir une fonction *saisir_PILE* en respectant le prototype : *PILE saisir_PILE()*. Cette fonction permet de saisir une pile en demandant à l'utilisateur d'entrer les entiers un par un et en les insèrant dans la pile.

   > Tester, AU FUR et À MESURE, TOUTES les fonctions en envisageant TOUS LES CAS possibles et sans JAMAIS écraser les tests déjà réalisés.

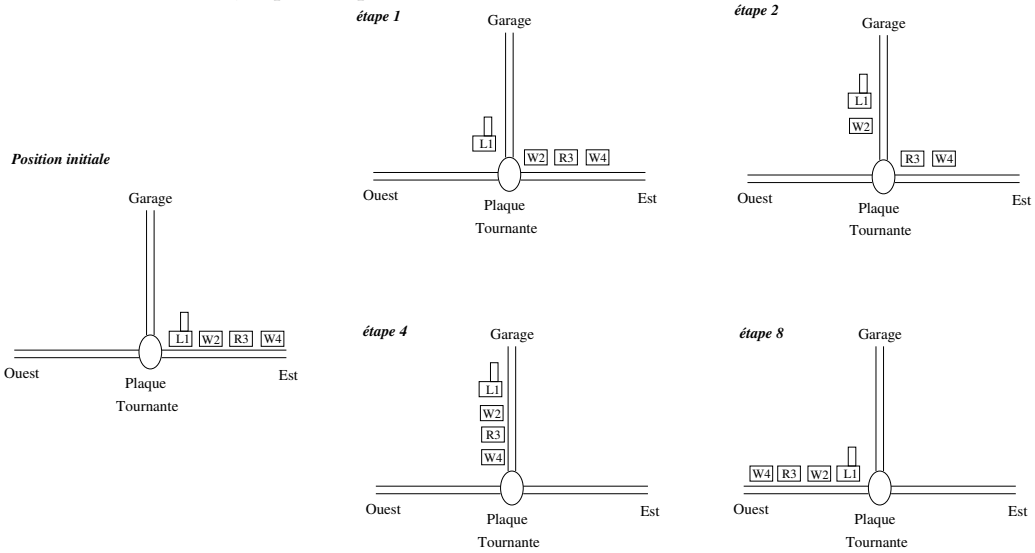
10. Une fois les tests effectués, transformer le fichier `pile_statique_P1.c` en unité `pile_statique` (décrites par les fichiers `pile_statique.h`, `pile_statique.c`, `tst_pile_statique.c`, voir la section 4 du support de cours sur les "Compléments au langage C").
    Télécharger le fichier `Makefile` disponible sur Moodle et le placer dans le répertoire où se trouve l'unité `pile_statique` (`TP_Pile_Statique_P1`).
    Dans ce même répertoire, exécuter depuis un shell la commande `make test1`. Le résultat doit être exactement le même que celui obtenu AVANT la transformation en unité.
11. Toujours dans le même répertoire, télécharger le fichier `progTestPileStat.c` disponible sur Moodle qui contient un programme de test qu'il est interdit de modifier.
    Modifier MAX afin qu'il soit égal à 100. Puis, dans ce même répertoire, exécuter depuis un shell la commande `make test2`. Le résultat à l'écran doit ne faire apparaître QUE la trace correcte de la compilation et des "OK".
12. Déposer sur Moodle une archive contenant la totalité du répertoire `TP_Pile_Statique_P1`.

Deuxième partie : pile statique d'éléments

1. Créer un répertoire `TP_Pile_Statique_P2`, s'y placer, y recopier l'unité `pile_statique` et créer un fichier `element_P2.c`.
2. Définir le type *ELEMENT* (pour les tests, vous prendrez un type *ELEMENT* défini sur le type *float*).
3. Ecrire toutes les fonctions permettant de manipuler des données du type *ELEMENT* :
   affiche_ELEMENT, *saisir_ELEMENT*, *affect_ELEMENT*, *compare_ELEMENT*.
   Tester ces fonctions.
4. Transformer le fichier `element_P2.c` de façon à créer l'unité `element` (décrite par les fichiers `element.h`, `element.c`, `tst_element.c`).
5. Modifier votre unité `pile_statique` de manière à utiliser des *ELEMENT* à la place des *int* dans la pile. ATTENTION : TOUTES les opérations sur les éléments de la pile ne devront plus se faire qu'à l'aide des fonctions définies dans l'unite `element`.
6. Déposer sur Moodle une archive contenant la totalité du répertoire `TP_Pile_Statique_P2`.

Troisième partie : application

Une plaque tournante est un mécanisme qui permet de faire faire demi-tour à un train. La plaque tournante est à l'intersection de 3 voies de chemin de fer : les voies Est, Ouest et la voie de garage. La plaque tournante peut prendre un élément (wagon ou locomotive) provenant d'une voie et le faire passer sur une autre voie. (elle ne peut prendre qu'un élément à la fois, elle prend le premier élément qui se présente et ne peut pas sauter d'élément). Un train est en général composé d'une locomotive en tête du train et de wagons, on disposera de wagons simples et de wagons restaurants.
Si on veut faire faire demi-tour à un train qui arrive de l'Est, il faut commencer par faire passer la locomotive sur la voie de garage, puis un à un tous les wagons du train, une fois tout le train passé sur la voie de garage, il faut le faire passer en prenant les wagons un à un, puis la locomotive sur la voie Ouest. Ainsi le train est totalement retourné, il peut repartir.



1. Créer un répertoire `TP_Pile_Statique_P3`, s'y placer, y recopier les unités `pile_statique` et `element` mises à jour en partie 2.
2. Définir en C la structure de donnée VOIE_DE_WAGONS qui représentera une voie (Est, Ouest ou Garage). Les composants d'un train devront être décrits par leur catégorie : Wagon simple (W), Wagon restaurant (R) ou Locomotive (L), et par leur numéro d'identification (entier positif).
   Pour cela, vous exploiterez les liens qui existent entre une voie et une pile et entre un composant du train et un élément de pile.
3. Ecrire un programme en C qui, à partir de la description d'une voie sur laquelle il y a un train (saisie par l'utilisateur), fait effectuer un demi-tour à ce train.
   Ce programme doit montrer à l'utilisateur les différentes étapes de ce demi-tour.
4. Déposer sur Moodle une archive contenant la totalité du répertoire `TP_Pile_Statique_P3`.