TP7 - Matrices et piles

Langage C (LC4)

1 Matrices

Question 1. Écrire une fonction `int** allocate_matrix(int lines, int columns, int value)` qui crée une matrice de taille `lines` × `columns` dont toutes les cases ont été initialisées à la valeur `value`.

Question 2. Écrire une fonction `void display_matrix(int lines, int columns, int** matrix)` qui permet d'affiche la matrice `lines` × `columns` `matrix`.

Question 3. Écrire une fonction `void free_matrix(int lines, int columns, int** matrix)` qui libère la matrice `matrix`. Les deux premiers arguments de la fonctions sont-ils réellement utiles ?

Question 4. Écrire une fonction `int** identity_matrix(int n)` qui génère la matrice identité de taille `n`.

Question 5. Écrire une fonction `int** sum_matrices(int l1, int c1, int** m1, int l2, int c2, int** m2)` qui calcule la somme matricielle des matrices `m1` et `m2` si leurs tailles sont compatibles et `NULL` sinon.

Question 6. Écrire une fonction `int** multiply_matrices(int l1, int c1, int** m1, int l2, int c2, int** m2)` qui calcule le produit matriciel des matrices `m1` et `m2` si leurs tailles sont compatibles et `NULL` sinon.

Question 7. Écrire une fonction `int** random_matrix(int lines, int columns)` qui génère une matrice aléatoire de taille `lines` × `columns`, en utilisant la fonction `rand`, dont la racine peut être initialisée de la façon suivante : `srand((int)time(NULL))`.

2 Piles

Une pile est une structure de données dans laquelle les derniers éléments ajoutés sont les premiers à être récupérés. Pour la représenter, on utilise la structure suivante, utilisant la structure de données de liste chaînée :

```
typedef struct element {
  int value;
  struct element* previous;
}* stack;
```

Question 8. Écrire une fonction `stack allocate_stack()` qui crée une pile vide.

Question 9. Écrire une fonction `void free_stack(stack s)` qui libère la mémoire occupée par la pile `s`.

Question 10. Écrire une fonction `int is_empty_stack(stack s)` qui renvoie un booléen et qui teste si la pile est vide.

Question 11. Écrire une fonction `void push_stack(stack s, int n)` qui empile l'entier `n` sur la pile `s`.

Question 12. Écrire une fonction `int pop_stack(stack s)` qui dépile la pile `p` et renvoie l'élément dépilé.

Question 13. Utiliser la structure de donnée de pile pour inverser l'ordre des éléments d'une liste donnée.