



Filière : 1 <sup>ère</sup> LGL	Matière : ASD 2 & Complexité		Enseignant : Dr. Sakka Rouïs
Date : 24/05 / 2025	Nbr de Crédits : 7	Coefficient : 1.5	Documents autorisés : Non
Durée de l'examen : 1h30	Régime d'évaluation : Mixte EX (70%) + DS (20%) + OR (10%)		Nombre de pages : 06
Nom & Prénom : .....			Matricule : .....
Signature :	Code confidentiel :		

**Exercice 1 : (6 Points)**

Imaginons que vous devez monter un escalier composé de  $n$  marches. Vous pouvez prendre un pas de 1 marche, un pas de 2 marches, ou un pas de 3 marches à chaque étape. L'objectif est de déterminer le **nombre de façons différentes** de monter cet escalier en utilisant ces 3 types de pas. Soit  $NBF(n)$  le nombre de façons de monter cet escalier de  $n$  marches. On peut remarquer que :

$NBF(0)=1$  (Il n'y a qu'une seule façon de ne pas monter, c'est rester au sol)

$NBF(1)=1$  (Il n'y a qu'une seule façon de monter 1 marche, c'est de prendre un pas de 1)

$NBF(2)=2$  (Il n'y deux façons : 1+1 et 2)

$NBF(n) = NBF(n-1) + NBF(n-2) + NBF(n-3)$  si  $n > 2$

1. Écrire une solution récursive pour la fonction suivante. (2 pts)

unsigned NBF (unsigned n) {

}

Donner la complexité temporelle de cette solution (sans justification). (2 pts)

*Ne rien écrire ici*

2. Proposer une seconde solution ayant une complexité temporelle en  $O(n)$ . (2 pts)

```
unsigned NBF (unsigned n){  
    int T[n];
```

```
}
```

---

**Annexe pour Exercice2 : fichier Liste.h**

---

```
struct cellule {  
    int cle;  
    struct cellule *suivant;    };  
struct liste {  
    struct cellule *premier;  
    struct cellule *derr    };  
void creerListe(struct liste *ll);  
unsigned listeVide(struct liste *ll);  
void insererApresElem(int x, struct cellule *p);  
void insererAvantElem(int x, struct cellule *p);  
void insererAvantPremier(int x, struct liste *ll);  
void insererApresDernier(int x, struct liste *ll);  
void supprimerElem(struct cellule *p);  
void supprimerPremier(struct liste *ll);  
void supprimerDernier(struct liste *ll);  
struct cellule *chercher(x, struct cellule *p);
```

## Exercice 2 : (14 Points)

On souhaite simuler un jeu entre deux joueurs. Chaque joueur possède une main, représentée par une liste chaînée, composée de cartes colorées : Rouge, Vert et Bleu. Le principe du jeu est le suivant :

- À tour de rôle, chaque joueur effectue les actions suivantes :
  1. Tirer une carte depuis le jeu, de manière aléatoire.
  2. Insérer la carte dans sa main selon les règles suivantes :
    - Si une carte de même couleur est présente, insérer la nouvelle carte juste avant celle-ci.
    - Sinon, ajouter la carte au début de la liste.
  3. Si un groupe de 3 cartes consécutives de même couleur est formé, il explose (les cartes sont supprimées de la main).
- Le jeu se termine lorsqu'un joueur n'a plus aucune carte dans sa main : il est déclaré vainqueur.

Pour implémenter ce jeu, on va modéliser la carte Rouge par 1, Vert par 2 et Bleu par 3.

### Illustration d'une itération de tirage :

Si la carte tirée par le joueur 1 est rouge (1) et que sa main contient la liste suivante :

L1 = 2 1 1 3    alors :

⇒ Insérer 1 dans la liste :      L1 = 2 1 1 1 3

⇒ Explosion du groupe des 1 :    L1 = 2 3

**Question : En important le fichier Liste.h vu en cours (voir Annexe de la page 2), on vous demande de compléter l'implémentation de ce jeu.**

```
#include "Liste.h" /* utiliser directement les opérations de l'annexe */
#include <stdio.h>
#include <stdlib.h>
#include <time.h> /* pour srand et time */
```

```
static struct liste L1;
static struct liste L2;
```

```
int tirerCarte ( ) {
    return 1 + rand() % 3; /* valeurs entre 1 et 3 inclus */
}
```

```
void initialiserJeu ( ) { /* 2 points */
```

```
    int carte;
```

```
    ..
```

```
    carte = urerCarte ( );
```

```
    insererAvantPremier(carte, .. ); /* Insère une première carte dans L1 */
```

```
    carte = tirerCarte ( );
```

```
    insererAvantPremier(carte, .. ); /* Insère une première carte dans L2 */
```

```
}
```

```
struct cellule* insererCarte (struct liste* L, int carte) { /*5 Points */
```

```
    struct cellule* p = L->premier;
```

```
void explosion (struct liste *L, struct cellule *pos){ /* 5 Points */
```

/\* À partir de la position 'pos', cette fonction supprime un groupe de 3 cartes consécutives ayant la même couleur que celle de 'pos', si un tel groupe existe. Sinon, elle ne fait rien. \*/

/\* Gérer tous les cas : début de liste ou suppression en fin de liste... \*/

```
/* Gérer tous les cas : début de liste ou suppression en fin de liste... */
```

3

```

unsigned commencerJeu ( ) { /*2 Points */
    int tour = 0;
    int carte;
    struct cellule *p;

    while (!listeVide( ..... ) && !listeVide( ..... )) {
        carte = tirerCarte();
        if (tour % 2 == 0) {
            p = insererCarte( ..... , carte); /* Insère la carte dans L1 */
            explosion( ..... , p); /* Tente une explosion dans L1 à partir de la cellule p */
        } else {
            p = insererCarte( ..... , carte); /* Insère la carte dans L1 */
            explosion( ..... , p); /* Tente une explosion dans L2 à partir de la cellule p */
        }
        tour++;
    }
    return listeVide (&L1) ? 1 : 2;
}

void main () {
    srand(time(NULL)); /* Initialiser le générateur aléatoire */
    initialiserJeu();
    int gagnant = commencerJeu();
    printf("Le gagnant est le joueur numéro %d \n", gagnant);
}

```