

Examen, INFO505

Durée : 1h30

Documents autorisés : tous documents du cours/td/tp, notes manuscrites (nb : pas de livres)

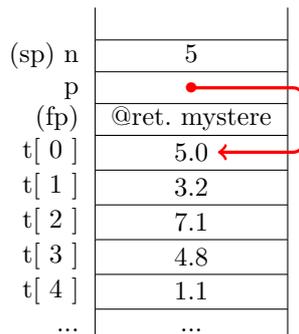
Les exercices sont indépendants. Le barème est indicatif. La durée est de 2h pour les étudiants bénéficiant d'un 1/3 temps.

1 Connaissance du C et pile d'exécution (/6)

On vous donne le programme C suivant :

```
void mystere( double* p, int n )
{
    // (1) début mystère
    for ( double* q = p + 1; q != p + n; q++ ) {
        double v = *q;
        double* r = q;
        // (2) 3eme fois ici
        for ( ; ( r != p ) && ( v < *(r-1) ); r-- )
            *r = *(r-1);
        *r = v;
    }
    // (3) fin mystère
}

int main()
{
    double t[5] = { 5.0, 3.2,
                  7.1, 4.8, 1.1 };
    mystere( t, 5 );
    for( int i = 0; i < 5; i++ )
        printf( "%f ", t[i] );
    printf( "\n" );
    return 0;
}
```



1. (/2,5) Que contient le tableau **t** à la fin ? Plus généralement, que fait la fonction **mystere** ?
2. (/3,5) On montre l'état de la pile d'exécution à la ligne (1) (sp : stack pointer, fp : frame pointer). Dessinez les états de la pile d'exécution aux étapes (2) et (3), en suivant la même convention.

2 Met en majuscule et enlève les autres symboles (/5)

Ecrivez une fonction `char* only_uppercase(char* p)` qui construit à partir d'une chaîne de caractères **p** une nouvelle chaîne de caractères, qui ne garde que les lettres de l'alphabet et les convertit en majuscule si nécessaire. Par exemple, le code suivant doit compiler :

```
int main()
{
    char* p1 = only_uppercase( "Bonjour Toto !" );
    char* p2 = only_uppercase( "Cogito Ergo Sum ? <dEsCaRtEs>" );
    printf( "%s %s\n", p1, p2 );
    free( p1 );
    free( p2 );
}
```

et affiche : **BONJOURTOTO COGITOERGOSUMDESCARTES**

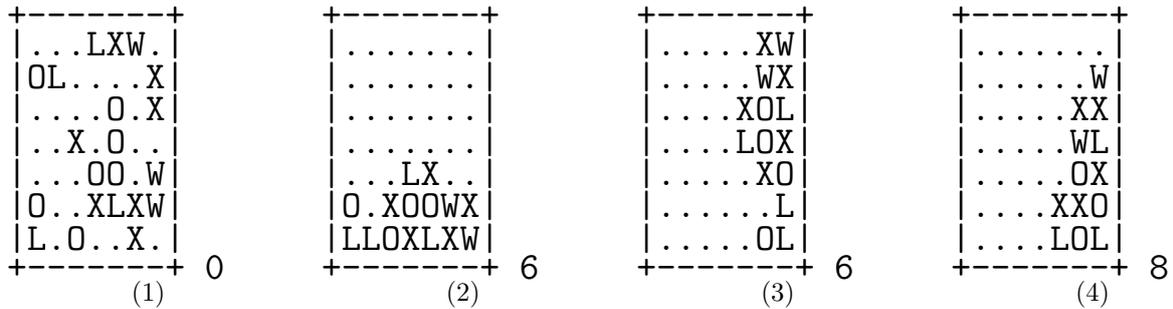


FIGURE 1 – Différentes étapes du jeu sont affichées (le score est en bas à droite). 1 : initialisation, 2 : après chute, 3 : après tourneCCW, 4 : après chute.

3 Jeu “Shake-it” (/12)

On se propose d’implémenter un petit jeu, dont le principe est simple. Dans une grille carrée, on dispose aléatoirement 4 types de jetons (ici ‘X’, ‘O’, ‘L’, ‘W’), en laissant environ une moitié de cases vide (‘.’). La gravité fait tomber les jetons (fonction `chute`). Quand deux jetons identiques tombent l’un sur l’autre, un seul reste, sinon les jetons différents s’empilent normalement (voir Figure, étapes (1) et (2)). Le joueur peut ensuite tourner sa grille dans le sens trigonométrique (fonction `tourneCCW`) ou dans le sens horaire (fonction `tourneCW`). Par exemple, l’étape (3) montre la grille après appel de `tourneCCW`, puis l’étape (4) après appel de `chute`. Le joueur gagne des points à chaque jeton qui disparaît, et gagne la partie si à la fin il n’y a plus que 4 jetons différents.

On définit la structure `Jeu` ainsi :

```
#define N 7
typedef char Grille[ N ][ N ];
struct SJeux {
    Grille g;
    int score;
};
typedef struct SJeux Jeux;
```

- (/0,5) Expliquez en 1 ligne pourquoi on utilise un “`#define N 7`” plutôt qu’un “`int N = 7;`” dans les lignes ci-dessus.
- (/0,5) Si j’écris la ligne “`Jeu J;`”, combien d’octets occupe la variable `J` en mémoire ?
- (/0,5) Si j’écris les lignes :

```
Jeu* pJeu;
pJeu->score = 0;
```

Est-ce que le code précédent compile ? Est-ce qu’il s’exécute correctement ? Justifiez brièvement.

- (/2) Ecrire la fonction “`void init(Jeux* pJeu, char* jetons)`” qui initialise le jeu avec environ une case sur 2 avec un jeton, et les jetons possibles choisis aléatoirement de manière uniforme parmi `jetons`. On peut l’appeler ainsi “`init(&J, "XOLW");`”, ce qui donne l’étape 1. On rappelle que “`rand() % n`” fabrique un entier aléatoire entre 0 et `n-1` inclus.
- (/3) Ecrire la fonction “`void chuteColonne(Jeux* pJeu, int j)`” qui fait tomber les jetons de la colonne `j` spécifiée (ce qui donne l’étape 2 si on l’appelle sur toutes les colonnes). Attention à bien éliminer les jetons identiques qui tombent les uns sur les autres. Le score est augmenté de 1 à chaque disparition.
- (/1) Ecrire la fonction “`void chute(Jeux* pJeu)`” qui fait chuter les jetons de toutes les colonnes.
- (/2) Ecrire (au choix) la fonction “`void tourneCCW(Jeux* pJeu)`” qui tourne la grille de 90 dans le sens trigonométrique (étape (3) de la figure), ou la fonction “`void tourneCW(Jeux* pJeu)`” qui tourne la grille de 90 dans le sens horaire.
- (/2,5) Ecrire une fonction “`int victoire(Jeux* pJeu, char* jetons)`” qui retourne vrai ssi il reste au maximum un jeton de chaque catégorie (il est possible d’avoir 0 jetons d’un type donné si aucun jeton de ce type n’avait été placé dans la grille au début).

Pour terminer le jeu chez vous, il suffit d’écrire la fonction d’affichage et un main qui demande au joueur s’il veut tourner à gauche ou à droite et teste si on gagne. On quitte lorsque le joueur a tourné 4 fois de suite sans faire augmenter le score.