Université Savoie Mont Blanc, UFR SceM
2021-2022

INFO505, L3 INFO
24 novembre 2021

Examen, INFO505

Durée : 1h30

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

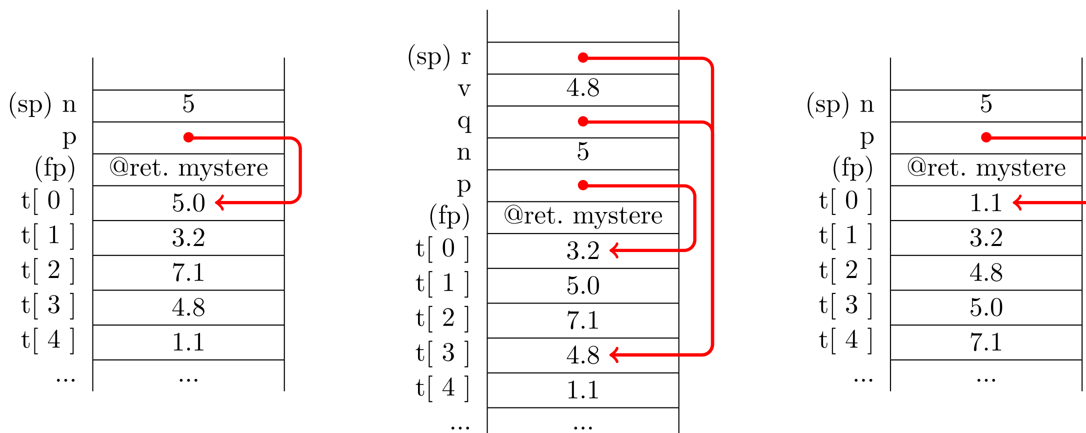
Les exercices sont indépendants. Le barème est indicatif. La durée est de 2h pour les étudiants bénéficiant d'un 1/3 temps.

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

On vous donne le programme C suivant :

```c
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
}
```

```c
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` ?

   Le programme affiche : `1.1 3.2 4.8 5.0 7.1`
   La fonction mystère trie les n valeurs flottantes situées à partir de l'adresse spécifiées par p. On reconnait un tri par insertion, car chaque valeur est insérée successivement en décalant les valeurs plus grandes (et dit autrement, si les données sont déjà dans l'ordre, chaque valeur est mise à la bonne place directement).

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.

   Voir ci-dessus.

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 :

```c
int main()
{
  char* p1 = only_uppercase( "Bonjour Toto !" );
```

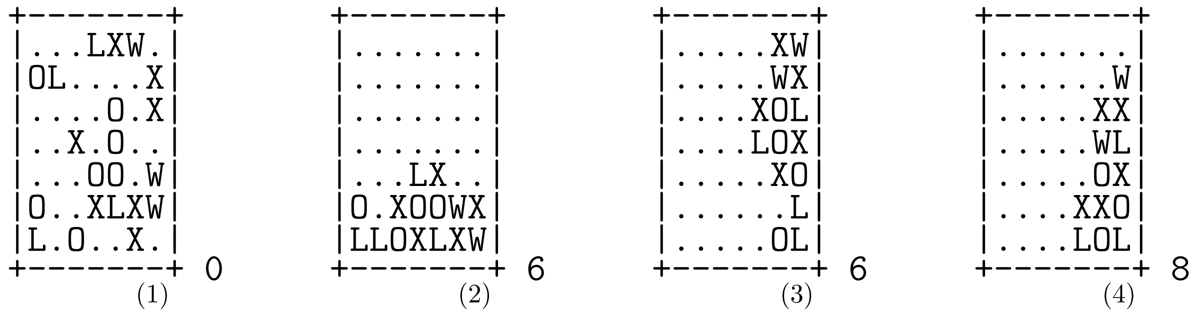


Figure 1 – Différentes étapes du jeu sont affichées (le score est en bas à droite). 1 : initialisation, 2 : après `chute`, 3 : après `tourneCCW`, 4 : après `chute`.

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

et affiche : `BONJOURTOTO COGITOERGOSUMDESCARTES`

```
/* Permet de mesurer la taille de la chaîne de caractères à allouer. */
int nb_lettres( char* p )
{
  int n = 0 ;
  for ( ; *p != 0 ; p++ )
    {
      char c = *p ;
      if ( ( 'a' <= c && c <= 'z' ) || ( 'A' <= c && c <= 'Z' ) ) n++ ;
    }
  return n ;
}

char* only_uppercase( char* p )
{
  // On précalcule le nombre de lettres en sortie
  int    n = nb_lettres ( p ) ;
  // On alloue la chaîne de caractères en sortie (+1 pour le '\0' final)
  char*  m = (char*) malloc( (n+1)*sizeof(char) ) ;
  char*  q = m ;            // q est la tête d'écriture
  for ( ; *p != 0 ; p++ ) // p est la tête de lecture
    {
      char c = *p ;
      if ( 'a' <= c && c <= 'z' )       *q++ = c + 'A' - 'a' ;
      else if ( 'A' <= c && c <= 'Z' ) *q++ = c ;
    }
  *q = 0 ; // on met bien le 0 ou '\0' final
  return m ;
}
```

3 Jeu "Shake-it" (/12)

On se propose d'implémenter un petit jeu, dont le principe est simple. Dans une grille carrée, on dispose alétoirement 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 disparait, et gagne la partie si à la fin il n'y a plus que 4 jetons différents.

On définit la structure `Jeu` ainsi :

```c
#define N 7
typedef char Grille[ N ][ N ];
struct SJeu {
  Grille g;
  int score;
};
typedef struct SJeu Jeu;
```

1. (/0,5) Expliquez en 1 ligne pourquoi on utilise un "`#define N 7`" plutôt qu'un "`int N = 7 ;`" dans les lignes ci-dessus.

   > Les types sont fixés à la compilation en C. Donc on ne peut définir ainsi un type grille où N serait choisi à l'exécution. Ici "`#define N 7`" est une directive préprocesseur, N sera remplacé par 7 dans le code **avant** compilation.

2. (/0,5) Si j'écris la ligne "Jeu J ;", combien d'octets occupe la variable `J` en mémoire ?

   > Cela instancie une variable de type `Jeu`. Celle-ci se décompose en 7×7 variables de type `char` (soit 1 octet chacune) et une variable de type `int` (le score, soit 4 octets), donc $7 \times 7 + 4 = 53$ octets. Pour des raisons d'alignement mémoire, 54 ou 56 octets seront associés à cette variable en pratique.

3. (/0,5) Si j'écris les lignes :

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

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

   > Le code compile. Mais il risque de faire une erreur de segmentation à l'exécution, car `pJeu` pointe à un endroit aléatoire en mémoire. Donc la variable `pJeu->score` a toutes les chances de vivre dans une zone invalide et non accessible par le processus. Sa modification à zéro provoquera une erreur de segmentation.

4. (/2) Ecrire la fonction "**void** init(Jeu* 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.

   ```c
   void init( Jeu* pJeu, char* symbols )
   {
     int n = strlen( symbols ) ;
     for ( int i = 0 ; i < N ; i++ )
       for ( int j = 0 ; j < N ; j++ )
         pJeu->g[ i ][ j ] = rand() % 2 ? '.'  : symbols[ rand() % n ] ;
     pJeu->score = 0 ;
   }
   ```

5. (/3) Ecrire la fonction "**void** chuteColonne(Jeu* 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 sur les uns sur les autres. Le score est augmenté de 1 à chaque disparition.

   ```c
   void chuteColonne( Jeu* pJeu, int j )
   {
     int w = N-1 ;
     for ( int i = N-2 ; i >= 0 ; i-- )
       {
         char c = pJeu->g[ i ][ j ] ;
         if ( c  != '.' )
           {
             if ( pJeu->g[ w ][ j ] == '.' )
               pJeu->g[ w ][ j ] = c ;
             else if ( pJeu->g[ w ][ j ]  != c )
               pJeu->g[ --w ][ j ] = c ;
             else pJeu->score += 1 ;
           }
       }
     for ( w-- ; w >= 0 ; w-- ) pJeu->g[ w ][ j ] = '.' ;
   }
   ```

6. (/1) Ecrire la fonction “**void** chute(Jeu∗ pJeu)” qui fait chuter les jetons de toutes les colonnes.

```
void chute( Jeu* pJeu )
{
  for ( int j = 0 ; j < N ; j++ )
    chuteColonne( pJeu, j ) ;
}
```

7. (/2) Ecrire (au choix) la fonction “**void** tourneCCW(Jeu∗ pJeu)” qui tourne la grille de 90 dans le sens trigonométrique (étape (3) de la figure), ou la fonction “**void** tourneCW(Jeu∗ pJeu)” qui tourne la grille de 90 dans le sens horaire.

```
// Au choix, tourneCW ou tourne CCW
void tourneCW( Jeu* pJeu )
{
  Jeu tmp = *pJeu ;
  for ( int i = 0 ; i < N ; i++ )
    for ( int j = 0 ; j < N ; j++ )
      pJeu->g[ i ][ j ] = tmp.g[ j ][ N - 1 - i ] ;
}
void tourneCCW( Jeu* pJeu )
{
  Jeu tmp = *pJeu ;
  for ( int i = 0 ; i < N ; i++ )
    for ( int j = 0 ; j < N ; j++ )
      pJeu->g[ i ][ j ] = tmp.g[ N - 1 - j ][ i ] ;
}
```

8. (/2,5) Ecrire une fonction “**int** victoire (Jeu∗ 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).

```
int victoire( Jeu* pJeu, char* jetons )
{
  int        n = strlen( jetons ) ;
  int* counter = (int*) malloc( n * sizeof( int ) ) ;
  for ( int i = 0 ; i < n ; i++ ) counter[ i ] = 0 ;
  for ( int i = 0 ; i < N ; i++ )
    for ( int j = 0 ; j < N ; j++ )
      if ( pJeu->g[ i ][ j ]  != '.' )
        for ( int k = 0 ; k < n ; k++ )
          if ( jetons[ k ] == pJeu->g[ i ][ j ] ) {
            counter[ k ] += 1 ;
            if ( counter[ k ] >= 2 ) return 0 ;
          }
  return 1 ;
```

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 à tourner 4 fois de suite sans faire augmenter le score.