Université de Nice-Sophia Antipolis — POLYTECH
Elec3 — 2018–2019

Examen de Info C

Durée : 1h
Aucun document autorisé
Mobiles interdits Nom : Prénom :

▶ 1. Dans la fonction suivante, insérez les affirmations qui donnent sa sémantique et qui prouvent la validité de l'énoncé itératif.

```
// Antécédent: ...

// Conséquent: ...


int mystere(int n) {

  int i=0, j=1, s=0;

  // Invariant : ....


  while(i<=n) {


    // ...
    // ...


    s = s + j;


    // ...
    // ...
    j = j + 2;


    // ...
    // ...

    i = i + 1;
    // ...
    // ...


  }


  // ...
```



```
  return s;

}
```

```
// Antécédent: n ⩾ 0
// Conséquent: carre(n) = n²
int carre(int n) {
  assert(n>=0);
  int i=0, j=1, s=0;
  // s = i² et j = 2i + 1
  while(i<n) {
    // invariant : s = i² et j = 2i + 1 et i < n
    // s + j = i² + j = i² + 2i + 1 = (i + 1)²
    s = s + j;
    // s = (i + 1)² et j = 2i + 1 ⇒ j + 2 = 2(i + 1) + 1
    j = j + 2;
    // s = (i + 1)² et j = 2(i + 1) + 1
    i = i + 1;
    // s = i² et j = 2i + 1
  }
  // s = i² et i = n ⇒ s = n²
  return s;
}
```

La finitude de la boucle est garantie par la fonction strictement décroissante $n - i$ qui tend vers 0, valeur pour laquelle son prédicat d'achèvement sera vérifier.

.....

▶ 2. Écrivez la fonction symétrique qui teste si une matrice carrée $n \times n$ passée en paramètre est symétrique ou non par rapport à la diagonale principale. Par exemple, la matrice suivante est symétrique :

$$\begin{pmatrix} 1.1 & 2.5 & 3.2 \\ 2.5 & -8.9 & 7.1 \\ 3.2 & 7.1 & 9 \end{pmatrix}$$

L'en-tête de la fonction est le suivant :

```
// Antécédent : ....
// Conséquent : ....
int symetrique(int n, double mat[][n])
```

```
/*
 * Antécédent : mat matrice carré initialisée
 * Conséquent : renvoie 1 si la matrice est symétique et 0 sinon
 * Note : seule la demi-matrice inférieure (sans la diagonale) est parcourue.
 */
int symetrique(const int n, int mat[][n])
{
  for (int i=1; i<n; i++) {
    for (int j=0; j<i; j++)
      if (mat[i][j] != mat[j][i])
        // mat matrice non symétrique
        return 0;
    // ∀i, j ∈ [0, i], mat[i, j] = mat[j, i]
  }
  // ∀i, j ∈ [0, n − 1], mat[i, j] = mat[j, i]
  return 1;
}
```

▶ 3. Écrivez la fonction `lireReel` qui lit sur l'entrée standard caractère à caractère (à l'aide de la fonction `getchar()`, à l'exclusion de toute autre fonction), un réel (positif ou négatif) et qui renvoie la valeur réelle (de type **double**) que représente la suite de caractères. Un réel sera formé d'une partie entière, éventuellement suivie d'un point (.), et dans ce cas, obligatoirement suivie d'une partie décimale. On ne traitera pas les cas d'erreur. Quelques exemples valides :

```
0.123 +34.345 -567 -230.0999 77.0 +10000.1 -234.255865 123
```

```
/*
 * Rôle : renvoie le prochain réel lu sur l'entrée standard
 */
double lireReel(void) {
  int c, negatif=0;
  double n = 0;
  // sauter les éventuels espaces de tête
  while (isspace(c=getchar())) ;
  // c est un chiffre ou + ou -
  if (c=='+' || c=='-') {
    negatif = c == '-';
    c = getchar();
  }
  // c est un chiffre
  do {
    n = n * 10 + c - '0';
  } while (isdigit(c = getchar()));
  // c n'est pas un chiffre. Est-ce un '.' ?
  if (c=='.') {
    // calculer la partie décimale
    c = getchar();
    // c est un chiffre
    int d=1;
    do {
      //
      n = n * 10 + c - '0';
      d*=10;
    } while (isdigit(c = getchar()));
    n=n/d;
  }
  //
  return negatif ? -n : n;
}
```

On lit l'entrée standard (uniquement avec `getchar`, à l'exclusion de toute autre fonction) qui contient une suite de caractères 1, 2, −, 3 et 4, et uniquement ces 5 caractères. Dans cette suite, on souhaite reconnaître des sous-suites qui commencent par un ou plusieurs 12, suivis d'un nombre (> 0) de tirets (−), suivis par un ou plusieurs 34.

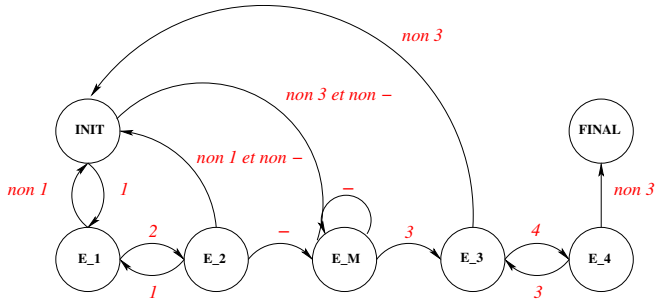
▶ 4. Écrivez un programme qui lit toute l'entrée standard et qui écrit sur la sortie standard le nombre de sous-suites, appelées motifs, reconnus bâtis sur le modèle précédent. Pensez à programmer un automate fini. Par exemple, votre programme écrira 2, si l'entrée standard contient :

```
121--341212-----343434--12-343412-3
```

les motifs reconnus sont en rouge ci-dessous :

```
121--341212-----343434--12-343412-3
```

L'automate qui représente les motifs à reconnaître est donné ci-dessous.



Sa programmation est simple et elle est donnée par la fonction `prochain_etat` qui donne toutes les transitions.

```
#include <stdio.h>
#include <stdlib.h>

enum ETAT { ETAT_INIT, ETAT_1, ETAT_2, ETAT_M, ETAT_3, ETAT_4, ETAT_FINAL };

/*
 * Rôle : fonction de transition qui renvoie le prochain
 *        état en fonction de l'état et du caractère courant
 */
enum ETAT prochain_etat(int c, enum ETAT etat) {
  switch (etat) {
      case  ETAT_INIT :
            if (c=='1') etat = ETAT_1;
            break;
      case  ETAT_1 :
            etat = (c=='2') ? ETAT_2 : ETAT_INIT;
            break;
      case  ETAT_2 :
            if (c=='1') etat = ETAT_1;
            else
              etat = (c=='-') ? etat = ETAT_M : ETAT_INIT;
            break;
      case  ETAT_M :
            if (c=='3') etat = ETAT_3;
            else
              if (c!='-') etat = ETAT_INIT;
            break;
      case  ETAT_3 :
            etat = (c=='4') ? ETAT_4 :  ETAT_INIT;
            break;
      case  ETAT_4 :
            if (c=='3') etat = ETAT_3;
            else etat =  ETAT_FINAL;
  }
  return etat;
}

int main(void) {
```

```
  enum ETAT etat = ETAT_INIT;
  int nbMotifs = 0, c, fini=0;

  while (!fini)
    if (etat == ETAT_FINAL) {
      // on a reconnu un nouveau motif
      etat =  ETAT_INIT;
      nbMotifs++;
    }
    else
      if ((c=getchar()) == EOF) fini=1;
      else
        // calculer le prochain état
        etat =  prochain_etat(c, etat);

  // EOF : on a parcouru toute l'entrée std
  printf("nbMotifs = %d\n", nbMotifs);

  return  EXIT_SUCCESS;
}
```