

Examen de Langage C (V. Granet)

Durée : 2h00

Aucun document autorisé
Mobiles interdits

- 1. On désire calculer la racine carrée d'un nombre réel x par la méthode de Heron¹. Pour cela, on calcule la suite $r_n = (r_{n-1} + x/r_{n-1})/2$ jusqu'à obtenir une approximation $r_n = \sqrt{x}$ telle que $r_n - x/r_n$ est inférieur à un ε donné. Vous pourrez choisir $r_0 = 1$. Écrivez en C la fonction `rac2` selon un algorithme itératif qui procède à ce calcul.

```
// algorithme trouvé par Héron d'Alexandrie au Ie siècle avant JC
// f(x) = 1/2(x + a/x)
// u0 = a > 0, u_{n+1} = f(u_n), u_n ≥ √a
// v_n = a/u_n
// e_n = u_n - v_n et 0 ≤ u_n - √a ≤ e_n
double rac2(double a) {
    const double epsilon = 1e-15;
    double r = 1;
    do
        r = (r + a/r)/2;
    while (r-a/r > epsilon);
    return r;
}
```

- 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(const int n, int mat[][n])

/*
 * Antécédent : mat matrice carré initialisée
 * Conséquent : renvoie 1 si la matrice est symétrique et 0 sinon
 * Note : seule la demi-matrice inférieure 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][i] != mat[j][i])
```

1. HÉRON L'ANCIEN, mathématicien et mécanicien grec du I^{er} siècle.

```
        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 (uniquement à l'aide de la fonction `scanf("%c", ...)`, 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, obligatoirement suivie d'un point (`.`), obligatoirement suivie d'une partie décimale. On ne traitera pas les cas d'erreur. Quelques exemples valides :

0.123 +34.345 -230.0999 77.0 +10000.1 -234.255865

```
/*
 * Rôle : renvoie le prochain caractère lu sur l'entrée standard
 */
int lireChar(void) {
    char c;
    int n = scanf("%c", &c);
    return n==EOF ? EOF : c;
}
/*
 * 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 = lireChar()));
    // c est un chiffre ou + ou -
    if (c=='+' || c=='-') {
        negatif = c == '-';
        c = lireChar();
    }
    // c est un chiffre
    do {
        // c est un chiffre
        n = n * 10 + c - '0';
    } while (isdigit(c = lireChar()));
    // c == '.'
    assert(c=='.');
    // calculer la partie décimale
    c = lireChar();
    // c est un chiffre
    int d=1;
    do {
        //
        n = n * 10 + c - '0';
        d*=10;
    } while (isdigit(c = lireChar()));
    n=n/d;
    //
    return negatif ? -n : n;
}
```

```

}

```

.....
 Un tableau de caractères contient une suite de lettres *a*, *b* et *c*, et uniquement ces 3 lettres.
 Dans cette suite, on souhaite reconnaître des sous-suites qui commencent par un ou plusieurs *a*,
 suivis d'un nombre impair de *b*, suivis par un ou plusieurs *c*.

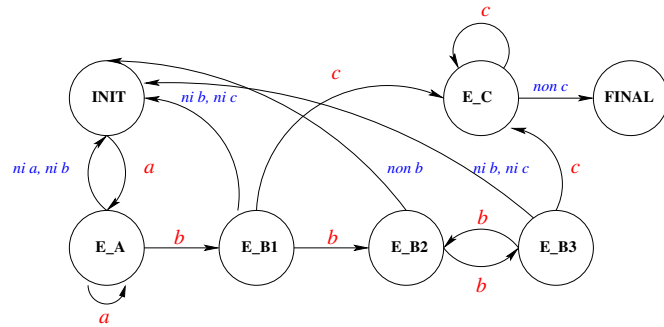
- 4. Écrivez un programme C qui parcourt tout le tableau et qui écrit sur la sortie standard le
 nombre de sous-suites, appelées motifs, bâtis sur le modèle précédent reconnues. Par exemple,
 votre programme écrira 3, si le tableau contient la suite de caractères :

cbbbaabbbbccbbcacabbcccabbbbaabbbbbbbcaabbccccbbccc

les motifs reconnus sont soulignés ci-dessous :

cbbbaabbbbccbbcacabbcccabbbbaabbbbbbbcaabbccccbbccc

Les motifs peuvent être reconnus simplement par l'automate donné ci-dessous.



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

```

/*
 * Automate pour reconnaître les motif  $a^+b^{2n+1}c^+$ 
 *
 * @author vg@unice.fr
 */
#include <stdio.h>
#include <stdlib.h>

enum ETAT { ETAT_INIT, ETAT_A, ETAT_B1, ETAT_B2, ETAT_B3, ETAT_C, 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=='a') etat = ETAT_A;
            break;
        case ETAT_A :

```

```

            if (c=='b') etat = ETAT_B1;
            else
                if (c!='a') etat = ETAT_INIT;
                break;
        case ETAT_B1 :
            if (c=='b') etat = ETAT_B2;
            else
                etat = (c=='c') ? ETAT_C : ETAT_INIT;
            break;
        case ETAT_B2 :
            etat = (c=='b') ? ETAT_B3 : ETAT_INIT;
            break;
        case ETAT_B3 :
            if (c=='c') etat = ETAT_C;
            etat = (c=='b') ? ETAT_B2 : ETAT_INIT;
        case ETAT_C :
            if (c!='c') etat = ETAT_FINAL;
            break;
    }
    return etat;
}

int main(void) {
    const int nbSymboles = 54;
    char input[] = "cbbbaabbbbccbbcacabbcccabbbbaabbbbbbbcaabbccccbbccc";
    enum ETAT etat = ETAT_INIT;
    int nbMotifs = 0, i=0;

    while (i<nbSymboles)
        if (etat == ETAT_FINAL) {
            // on a reconnu un nouveau motif
            etat = ETAT_INIT;
            nbMotifs++;
        }
        else
            // calculer le prochain état
            etat = prochain_etat(input[i++], etat);
    // on a parcouru tout le tableau
    printf("nbMotifs = %d\n", nbMotifs);

    return EXIT_SUCCESS;
}
.....

```