

Examen de Examen de Langage C

Durée : 1h30

Aucun document autorisé – Mobiles interdits

Notez que les affirmations (antécédents, conséquents, rôles, et invariants) dans vos codes C entreront pour partie dans la note finale.

1 Voyelles

- 1. Écrivez la fonction `estUneVoyelle` qui prend en paramètre un caractère et qui renvoie 1 si ce caractère est une voyelle et 0 sinon. Rappel : en français, il existe 6 voyelles : 'a','e','i','o','u','y' en minuscule et 'A','E','I','O','U','Y' en majuscule.

```
/*
 * Rôle : renvoie 1 si c est une voyelle et 0 sinon.
 */
int estUneVoyelle(const char c) {
    switch (tolower(c)) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'y': return 1;
    }
    // c n'est pas une voyelle
    return 0;
}
```

- 2. Écrivez la fonction `compterVoyelles` qui prend en paramètre un tableau de caractères `t` et qui renvoie le nombre de voyelles qu'il contient. Le nombre de caractères `n` contenus dans le tableau `t` et le tableau `t` sont passés en paramètres de la fonction.

```
/*
 * Antécédent : n>0
 * Rôle : renvoie le nombre de voyelles contenues dans t
 */
int compterVoyelles(const int n, const char t[]) {
    int nbVoyelles=0;
    for (int i=0; i<n; i++)
        if (estUneVoyelle(t[i]))
            // le caractère courant est une voyelle
            nbVoyelles++;
}
```

```
    // on a compté toutes les voyelles
    return nbVoyelles;
}
```

- 3. En utilisant la fonction `compterVoyelles` précédente, écrivez la fonction `main` qui écrit sur la sortie standard le nombre de voyelles contenues dans le mot *Anticonstitutionnellement*.

```
int main(void) {
    printf("nb voyelles : %d\n", compterVoyelles(25, "Anticonstitutionnellement"));
    return EXIT_SUCCESS;
}
```

2 Matrice

- 4. Écrivez la procédure `initAléa` qui initialise de façon aléatoire une matrice `mat` d'entiers (`int`) de dimension $m \times n$. Vous utiliserez la fonction `rand`.

```
/*
 * Antécédent : m>0 et n>0
 * Rôle : initialise de façon aléatoire la matrice mat
 */
void initAléa(const int m, const int n, int mat[][n]) {
    for (int i=0; i<m; i++)
        // initialiser la ligne i
        for (int j=0; j<n; j++)
            mat[i][j] = rand();
}
```

- 5. Écrivez la fonction booléenne `symétrique` qui teste si une matrice carrée `mat` d'entiers (`int`) de dimension $n \times n$ passée en paramètre est *symétrique* ou non par rapport à sa 1ère diagonale. **Attention de ne parcourir que le strict nécessaire.**

```
/*
 * Antécédent n>0
 * Rôle : renvoie 1 si la matrice mat est symétrique, et 0 sinon
 */
int symétrique(const int n, const int mat[][n]) {
    // on parcourt uniquement la demi-matrice inférieure
    // sous la diagonale
    for (int i=1; i<n; i++) {
        for (int j=0; j<i; j++)
            if (mat[i][j]!=mat[j][i])
                // non symétrique
                return 0;
    }
    return 1;
}
```

```

        return 0;
        // la sous-matrice mat[0..i; 0..i] est symétrique
    }
    // la matrice mat[0..n-1; 0..n-1] est symétrique
    return 1;
}

```

3 Matrice et Vecteur

- 6. Écrivez la procédure `vecteurSomme` qui prend en paramètre une matrice `mat` de dimension $m \times n$ et un vecteur `vect` de dimension m . Cette procédure affecte à chaque `vect[i]` la somme des valeurs entières de chaque ligne i de la matrice. Pensez à écrire une fonction `somme`.

```

/*
 * antécédent : n>0
 * rôle : renvoie la somme des entiers
 *         du tableau t
 */
int somme(const int n, const int t[]) {
    int som=0;
    for (int i=0; i<n; i++)
        som+=t[i];
    // som =  $\sum_{i=0}^n t[i]$ 
    return som;
}

/*
 * antécédent : m>0 et n>0 et mat initialisée
 * rôle : crée le vecteur v formé de la somme
 *         des entiers de chacune des lignes
 *         de la matrice mat
 */
void créerVecteur(const int m, const int n,
                  const int mat[][n], int vect[])
{
    for (int i=0; i<m; i++)
        // calculer la somme des entiers
        // de la ligne : mat[i]
        vect[i] = somme(n, mat[i]);
}

```