

Examen de Info 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. En C, quelles sont les valeurs possibles pour l'indice d'un tableau de n éléments ?

En C, les indices d'un tableau de n éléments sont des entiers pris sur l'intervalle $[0; n - 1]$.

- 2. Expliquez de façon claire et synthétique ce qui se produit en C à la compilation et à l'exécution des instructions suivantes :

```
int i=-1;
int x = t[i];
```

Bien évidemment, c'est une erreur de programmation, puisque l'indice -1 est faux. Mais, le compilateur ne signale aucune erreur et à l'exécution, soit l'adresse $t[-1]$ est licite et x prendra la valeur qui existe à cette adresse, soit l'adresse est illicite et le programme plantera (**segmentation fault**) ; ce qui est préférable.

- 3. En C, existe-t-il un type *chaîne de caractères* ? Expliquez de façon claire et synthétique comment sont représentées en C les *chaînes de caractères*.

En C, il n'existe pas de type *chaîne de caractères*. Toutefois, on peut manipuler des chaînes de caractères qui sont représentées par des tableaux de caractères. Le caractère `'\0'` indique la fin de la chaîne.

- 4. Écrivez en C la fonction booléenne `estIdentite` qui teste si une matrice carrée $n \times n$ passée en paramètre est une matrice identité ou non. **Attention : ne faire aucun test : $i=j$ (ou $i!=j$).**

```
/*
 * Antécédent : n>0 ordre de la matrice carrée
 * Conséquent : renvoie 1 si mat est une matrice identité et faux sinon
 */
int estIdentite(const int n, const int mat[][n]) {
```

```
    // vérifier que les deux 1/2 matrices inférieure et supérieure
    // à l'exclusion de la diagonale ne contiennent que des 0
    // et que la diagonale contient des 1
    for (int i=0; i<n; i++) {
        // vérifier l'élément sur la diagonale
        if (mat[i][i]!=1) return 0;
        // vérifier les 2 éléments symétriques de la diagonale
        for (int j=0; j<i; j++)
            if (mat[i][j]!=0 || mat[j][i]!=0)
                // une valeur différente de 0 ⇒ matrice non identité
                return 0;
    }
    // c'est bien une matrice identité !
    return 1;
}
```

- 5. Écrivez en C la procédure produit qui calcule le produit de deux matrices d'entiers (`int`) $A(m, p)$ et $B(p, n)$. On rappelle que $C(m, n) = A(m, p) \times B(p, n)$, avec $C_{i,j} = \sum_{k=1}^p A_{i,k} \times B_{k,j}$.

```
/*
 * Antécédent : A[m][p] et B[p][n] deux matrices d'entiers initialisées
 * Conséquent : C[m][n] = A[m][p] × B[p][n]
 */
void produit(const int m, const int n, const int p,
             const int A[][p], const int B[][n], int C[][n])
{
    for (int i=0; i<m; i++)
        for (int j=0; j<n; j++) {
            // calculer les Ci,j
            int som = 0;
            for (int k=0; k<p; k++) som+=A[i][k]*B[k][j];
            // som = ∑k=1p Ai,k × Bk,j
            C[i][j] = som;
        }
}
```

- 6. Écrivez en C la fonction `main` qui recopie toute l'entrée standard sur la sortie standard en supprimant toutes les lignes vides. Les lectures et les écritures devront se faire **uniquement** avec les fonctions `getchar` et `putchar`. Exemple : ci-dessous, à gauche l'entrée standard (sans les numéros de lignes), à droite le résultat sur la sortie standard (sans les numéros de lignes) :

1	1 une ligne vide précède et deux suivent
2 une ligne vide précède et deux suivent	2 blabla blabla blabla
3	3 blabla blabla blabla
4	4 2 lignes vides suivent...
5 blabla blabla blabla	
6 blabla blabla blabla	
7 2 lignes vides suivent...	
8	
9	

```

int main (void) {
    int c, cpred='\n';
    // lire toute l'E/S
    while ((c=getchar())!=EOF) {
        if (c!='\n' || cpred!='\n')
            putchar(c);
        // sinon la ligne est vide => ne rien écrire
        // le car courant devient le précédent
        cpred=c;
    }
    return EXIT_SUCCESS;
}

```

```

    return 1;
}

```

Un *palindrome* est un mot (ou un groupe de mots séparés par des espaces) qui peut être lu indifféremment de gauche à droite ou de droite à gauche en conservant le même sens. Par exemple, *a*, *radar* et *elu par cette crapule* sont trois palindromes, mais pas *toto*, ni *elec3*.

- 7. Sans utiliser de fonctions de `string.h`, écrivez en C la fonction booléenne `palindrome` qui teste si une chaîne de caractères passée en paramètre est un palindrome ou non. Une chaîne vide ou ne contenant que des espaces **n'est pas** un palindrome. Pour tester un espace, vous pouvez utiliser la fonction `isspace`.

```

/*
 * Rôle : teste si le mot (ou groupe de mots séparés par des espaces)
 *        désigné par la chaîne de caractères p est un palindrome ou
 *        non
 */
int palindrome(const char p[])
{
    // trouver l'indice de fin de chaîne
    // et compter le nombre d'espaces
    int i=0, nbSpaces=0;
    while (p[i]!='\0') {
        if (isspace(p[i])) nbSpaces++;
        i++;
    }
    if (i==0 || nbSpaces==i)
        // chaîne vide ou ne contenant que des espaces
        // => ce n'est pas un palindrome
        return 0;
    // vérifier si p est un palindrome
    // en parcourant la chaîne par les deux extrémités
    // gauche et droite
    int gauche=0, droite=i-1;
    while (gauche<droite)
        // sauter les espaces si nécessaire
        if (isspace(p[gauche])) gauche++;
        else
            if (isspace(p[droite])) droite--;
        else
            // les caractères p[gauche] et p[droite] ne sont pas des espaces
            // tester s'ils sont identiques ou pas
            if (p[gauche++] != p[droite--]) return 0;
    // p est un palindrome

```