

# Algorithmes de carrés magiques [tb04]

## Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel  algoprog  Version 19 mai 2018

## Table des matières

<b>1 Algorithmes de construction / pgcmagique</b>	<b>2</b>
1.1 Représentation du Carré . . . . .	2
1.2 Utilitaires . . . . .	2
1.3 Premier algorithme . . . . .	3
1.4 Autre algorithme . . . . .	6
1.5 Troisième algorithme . . . . .	8
<b>2 Références générales</b>	<b>9</b>

## C++ - Algorithmes de construction (Solution)



**Mots-Clés** Tableau multidimensionnel, Carré magique ■

**Requis** Structures de base, Structures conditionnelles, Algorithmes paramétrés, Structures répétitives, Schéma itératif, Tableaux ■

**Difficulté** ●●○ (1 h) ■



### Objectif

Cet exercice réalise des algorithmes de construction de carrés magiques d'ordre impair  $n$ .

# 1 Algorithmes de construction / pgcmagique

## 1.1 Représentation du Carré



Définissez la constante `TMAX=100` (nombre maximal de cases d'un tableau), le type `ITableau` comme étant un tableau de `TMAX` entiers ainsi que le type `TCarre` comme étant équivalent au type `ITableau`.



Validez vos définitions avec la solution.

**Solution C++** @[pgcmagique.cpp]

```
/**
 * Taille maximale du tableau
 */
const int TMAX = 100;

/**
 * Tableau d'entiers
 */
using ITableau = int[TMAX];

/**
 * Représente un Carré
 */
using TCarre = ITableau;
```



Soient :

- La fonction `evalCase(t,j,k,n)` qui renvoie la valeur de l'élément en `(j,k)` d'un `ITableau t` de `n` colonnes.
- La procédure `fixerCase(t,j,k,n,valeur)` qui fixe l'élément en `(j,k)` d'un `ITableau t` de `n` colonnes à la valeur `valeur`.

**Attention**, les indices sont numérotés à partir de 1.

C++ @[evalCase/fixerCase] (dans UtilsTM)

## 1.2 Utilitaires



Écrivez une fonction `saisirOrdreCarre(nmax)` qui renvoie un entier, saisi par l'utilisateur, entier qui doit être impair, supérieur à 1 et tel que  $n^2$  soit inférieur ou égal à `nmax` (entier). Affichez l'invite :

Ordre impair du carré?



Validez votre fonction avec la solution.

**Solution C++** @[pgcmagique.cpp]

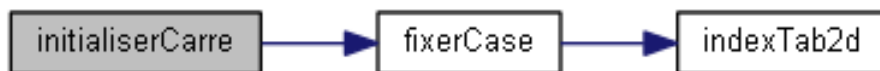
```

/**
 * Saisit l'ordre (impair) d'un TCarre
 * @param[in] nmax - nombre maximum de cellules du TCarre
 * @return l'ordre impair du TCarre
 */
int saisirOrdreCarre(int nmax)
{
    int n = 0;
    while (!(n >= 1 && n % 2 == 1 && n * n <= nmax))
    {
        cout<<"Ordre impair du carre? ";
        cin>>n;
    }
    return n;
}

```



Écrivez une procédure `initialiserCarre(t,n)` qui initialise à zéro chacun des éléments d'un ITableau `t` d'ordre `n`.



Validez votre procédure avec la solution.

**Solution C++** @[UtilsTM.cpp]

```

/**
 * Initialise à zéro un TCarre
 * @param[out] t - un ITableau
 * @param[in] n - ordre de t
 */
void initialiserCarre(ITableau& t, int n)
{
    for (int j = 1; j <= n; ++j)
    {
        for (int k = 1; k <= n; ++k)
        {
            fixerCase(t,j,k,n,0);
        }
    }
}

```

### 1.3 Premier algorithme

L'algorithme ci-après ne fonctionne que si  $n$  est **impair**.

**Algorithme**

Il place les nombres dans l'ordre  $1, 2, \dots, n^2$ .

- Placez un 1 au milieu de la première ligne.
- Après avoir placé  $k$  dans le carré en  $(i, j)$ , placez  $k + 1$  dans le carré à droite et vers le haut, en défilant au-delà des bordures, c.-à-d. que si la case dépasse la première ligne (resp. la dernière colonne), la case sélectionnée est celle en dernière ligne (resp. en première colonne). En cas d'occupation d'une case, la case suivante est recalculée sous la case courante. Par construction, on démontre que cette case n'est pas occupée.

**Exemple**

Voici le carré d'ordre 3 obtenu à l'aide de cette méthode.

8	1	6
3	5	7
4	9	2



Déroulez l'algorithme sur le carré d'ordre 3 et vérifiez que vous obtenez celui de l'exemple.

**Solution simple**

Voici le déroulement de l'algorithme avec les placements :

2a			7a			9a		
	1			1/4a	6	8b	1	6
			3b	5	7e	3	5	7
		2b	4b/7d	2		4	9b	2

**Commentaires**

(1) On place le nombre dans la case vide. (2a) La ligne est hors du carré : (2b) on repositionne le nombre sur la dernière ligne. (3a) La colonne est hors du carré : (3b) on le repositionne sur la première colonne. (4a) La case est occupée : on place le nombre sous la case courante. (5) On place le nombre dans la case vide. (6) On place le nombre dans la case vide. (7a) La ligne est hors du carré : (7b) on le repositionne sur la dernière ligne. (7c) La colonne est hors du carré : (7d) on le repositionne sur la première colonne ; la case est occupée par le nombre 4 : (7e) on place le nombre sous la case courante. (8a) La colonne est hors du carré : (8b) on le repositionne sur la première colonne. (9a) La ligne est hors du carré : (9b) on le repositionne sur la dernière ligne. FIN Tous les nombres de 1 à 9 ont été placés.



Écrivez une fonction `suivant(k,n)` qui renvoie le suivant de l'entier  $k$  dans l'ensemble circulaire  $[1..n]$ .



De même, écrivez une fonction `precedent(k,n)` qui renvoie le précédent de l'entier  $k$  dans l'ensemble circulaire  $[1..n]$ .



Validez vos fonctions avec la solution.

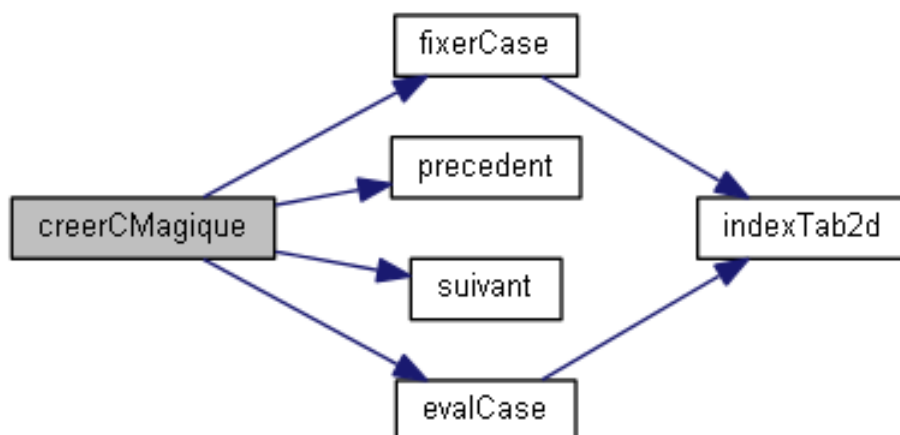
### Solution C++ @[pgcmagique.cpp]

```
/**
 * Suivant dans l'ensemble circulaire [1..n]
 * @param[in] k - un entier
 * @param[in] n - un entier
 * @return le suivant de k dans l'ensemble circulaire [1..n]
 */
int suivant(int k, int n)
{
    return (k >= n ? 1 : k + 1);
}

/**
 * Précédent dans l'ensemble circulaire [1..n]
 * @param[in] k - un entier
 * @param[in] n - un entier
 * @return le précédent de k dans l'ensemble circulaire [1..n]
 */
int precedent(int k, int n)
{
    return (k == 1 ? n : k - 1);
}
```



Écrivez alors une procédure `creerCMagique1(c,n)` qui crée le carré magique d'ordre `n` dans un `TCarre c` selon l'algorithme ci-dessus.



Validez votre procédure avec la solution.

### Solution C++ @[pgcmagique.cpp]

```
/**
 * Crée un carré magique
```

```

@param[out] c - un TCarre
@param[in] n - ordre de c
*/

void creerCMagique1(TCarre& c, int n)
{
    UtilsTM::initialiserCarre(c,n);
    int nxn = n * n;
    int ix = 1;
    int jx = n / 2 + 1;
    UtilsTM::fixerCase(c,ix,jx,n,1);
    for (int valeur = 2; valeur <= nxn; ++valeur)
    {
        int isucc = precedent(ix,n);
        int jsucc = suivant(jx,n);
        if (UtilsTM::evalCase(c,isucc,jsucc,n) == 0)
        {
            ix = isucc;
            jx = jsucc;
        }
        else
        {
            ix = suivant(ix,n);
        }
        UtilsTM::fixerCase(c,ix,jx,n,valeur);
    }
}

```



Écrivez un programme de sorte qu'il :

- Saisit l'ordre du carré dans un entier (par exemple *n*).
- Déclare un *TCarre* (par exemple *c*).
- Construit le *TCarre* magique d'ordre *n* dans *c*.
- Enfin affiche *c*.



Testez. Exemples d'exécution :

Ordre impair du carré? 3

```

8  1  6
3  5  7
4  9  2

```

Ordre impair du carré? 5

```

17 24  1  8 15
23  5  7 14 16
 4  6 13 20 22
10 12 19 21  3
11 18 25  2  9

```

## 1.4 Autre algorithme

Voici un autre algorithme de construction selon le même principe.

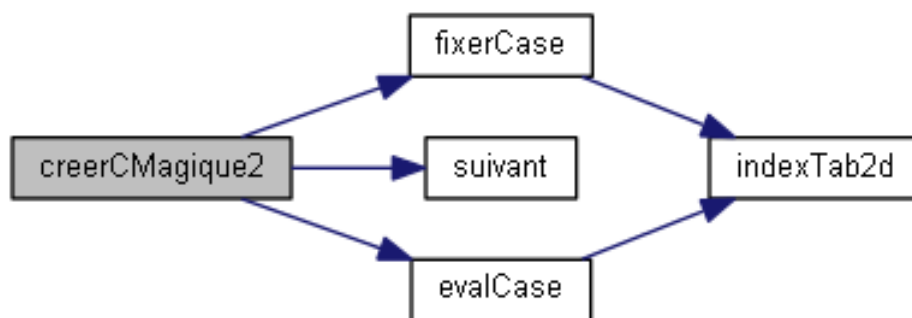
**Algorithme**

Il place les nombres dans l'ordre  $1, 2, \dots, n^2$ .

- Placez le 1 dans la case située une ligne en dessous de la case centrale.
- Après avoir placé  $k$  dans une case de coordonnées  $(i, j)$ , placez  $k + 1$  dans la case  $(\ell, c) = (i + 1, j + 1)$ , en défilant au-delà des bordures, c.-à-d. que si la case dépasse la dernière ligne (resp. la dernière colonne), la case sélectionnée est celle en première ligne (resp. en première colonne). En cas d'occupation d'une case, la case suivante est  $(\ell, c) = (i + 2, j)$ . Par construction, on démontre que cette case n'est pas occupée.



Écrivez une procédure `creerCMagique2(c,n)` qui crée le carré magique d'ordre  $n$  dans un `TCarre c` selon cet algorithme.



Validez votre procédure avec la solution.

**Solution C++**

@[pgcmagique.cpp]

```

/**
 * Crée un carré magique (méthode 2)
 * @param[out] c - un TCarre
 * @param[in] n - ordre de c
 */

void creerCMagique2(TCarre& c, int n)
{
    UtilsTM::initialiserCarre(c,n);
    int nxn = n * n;
    int ix = (n + 1) / 2 + 1;
    int jx = (n + 1) / 2;
    UtilsTM::fixerCase(c,ix,jx,n,1);
    for (int valeur = 2; valeur <= nxn; ++valeur)
    {
        int isucc = suivant(ix,n);
        int jsucc = suivant(jx,n);
        if (UtilsTM::evalCase(c,isucc,jsucc,n) == 0)
        {
            ix = isucc;
            jx = jsucc;
        }
    }
    else

```

```

    {
        ix = suivant(isucc,n);
    }
    UtilsTM::fixerCase(c,ix,jx,n,valeur);
}
}

```



Modifiez l'appel en celui de `creerCMagique2(c,n)` dans votre programme.



Testez. Exemple d'exécution :

```

Ordre impair du carré? 5
11 24  7 20  3
 4 12 25  8 16
17  5 13 21  9
10 18  1 14 22
23  6 19  2 15

```



Validez votre programme avec la solution.

**Solution C++**     @[pgcmagique.cpp]

```

int main()
{
    int n = saisirOrdreCarre(TMAX);
    TCarre c;

    cout<<"CMagique1"<<endl;
    creerCMagique1(c,n);
    UtilsTM::afficherCarre(c,n);

    cout<<"CMagique2"<<endl;
    creerCMagique2(c,n);
    UtilsTM::afficherCarre(c,n);
}

```

## 1.5 Troisième algorithme

Enfin voici un troisième algorithme qui est similaire à @[Autre algorithme].

### Algorithme

Il place les nombres dans l'ordre  $1, 2, \dots, n^2$ .

- Placez le 1 dans la case située une ligne en-dessous de la case centrale.
- Après avoir placé  $k$  dans une case de coordonnées  $(i, j)$ , placez  $k + 1$  dans la case  $(\ell, c) = (i + 1, j + 1)$ , en défilant au-delà des bordures, c.-à-d. que si la case dépasse la dernière ligne (resp. la dernière colonne), la case sélectionnée est celle en première



ligne (resp. en première colonne). **En cas d’occupation d’une case, on essaie alors répétitivement de placer le nombre en  $(\ell + 1, c - 1)$ , toujours en défilant au-delà des bordures.**

## 2 Références générales

Comprend [Maunoury-AL1 :c8 :ex12] ■