

The background features several overlapping, glowing loops in shades of teal and orange, resembling light trails or smoke. A semi-transparent white horizontal rectangle is centered across the middle of the image.

# Répétitions

# Objectifs

- Appréhender les répétitions
- Différencier les principales formes de répétitions
- Connaître les équivalences en C#

# Problème

- Écrire un algorithme qui affiche les nombres de 1 à 10
- Solution :

```
écrireNL(1);  
écrireNL(2);  
écrireNL(3);  
écrireNL(4);  
écrireNL(5);  
écrireNL(6);  
écrireNL(7);  
écrireNL(8);  
écrireNL(9);  
écrireNL(10);
```

# Problème

```
écrireNL(1);  
écrireNL(2);  
écrireNL(3);  
écrireNL(4);  
écrireNL(5);  
écrireNL(6);  
écrireNL(7);  
écrireNL(8);  
écrireNL(9);  
écrireNL(10);
```

- Quels sont les principaux problèmes ici ?
  - Qu'est-ce qui se passe si le client veut aller jusqu'à 100 ?
  - Comment puis-je faire pour demander à l'utilisateur jusqu'à combien il veut aller ?
  - Est-ce normal de répéter du code ? (Ex. risque d'erreur, lisibilité)

# Solution ?




Algorithme textuel :

- Lire la valeur maximale **nombreMax** à afficher
- Afficher les entiers de 1 à **nombreMax**

Comment traduire cela en pseudo-code ?

- Ajout de structures de contrôle de type répétition

# Répétition « tant que »

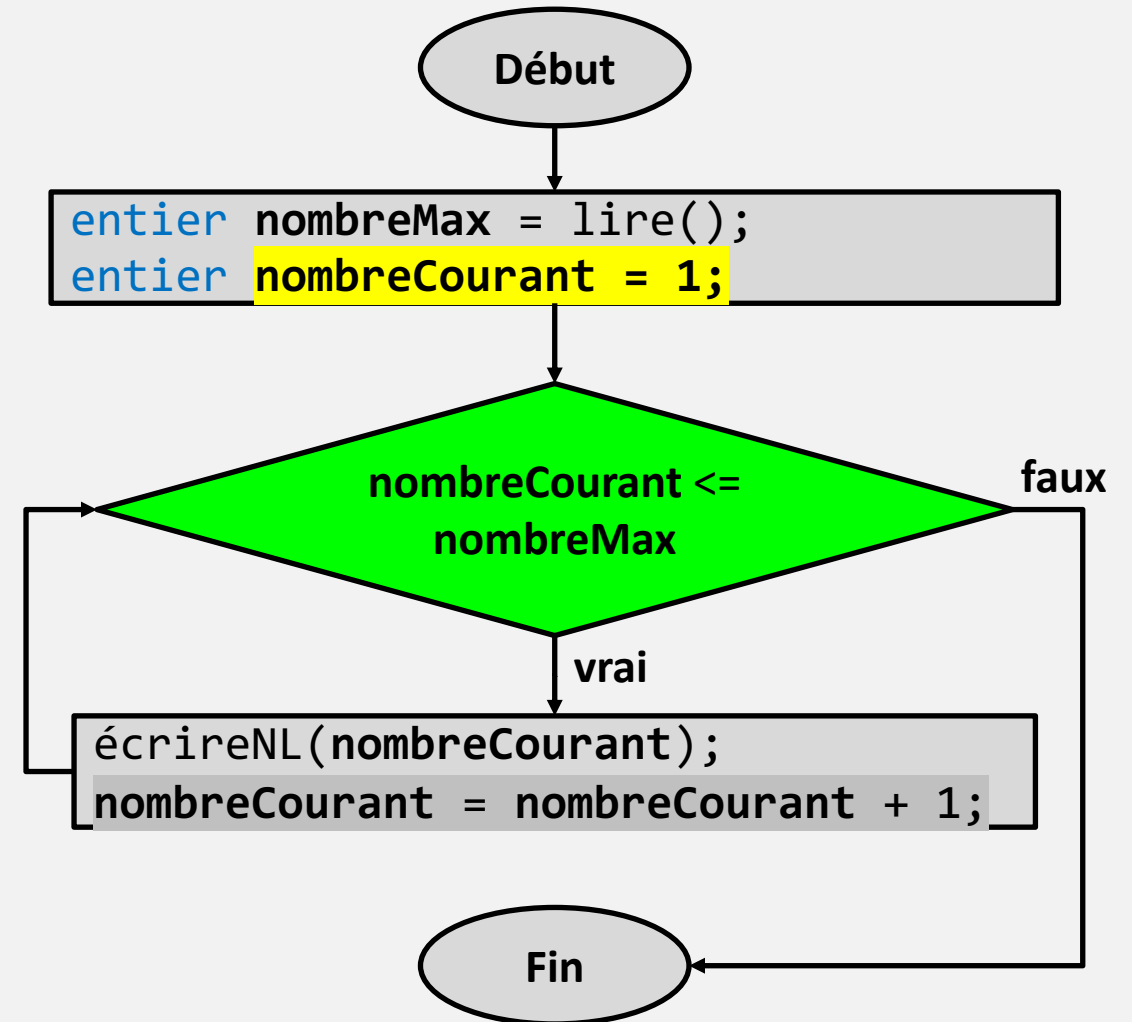
 : initialisation  
 : condition  
 : incrémentation

```
entier nombreMax = lire();  
entier nombreCourant = 1;  
tant que (nombreCourant <= nombreMax) faire {  
    écrireNL(nombreCourant);  
    nombreCourant = nombreCourant + 1;  
}
```




```
tant que <conditionEstVraie> faire {  
    <algorithmeARépéter>  
}
```

La condition est évaluée avant de rentrer dans la boucle

Ne pas oublier de faire évoluer ce qui est évalué comme condition  
sinon vous aurez une boucle infinie !



# Répétition « pour »

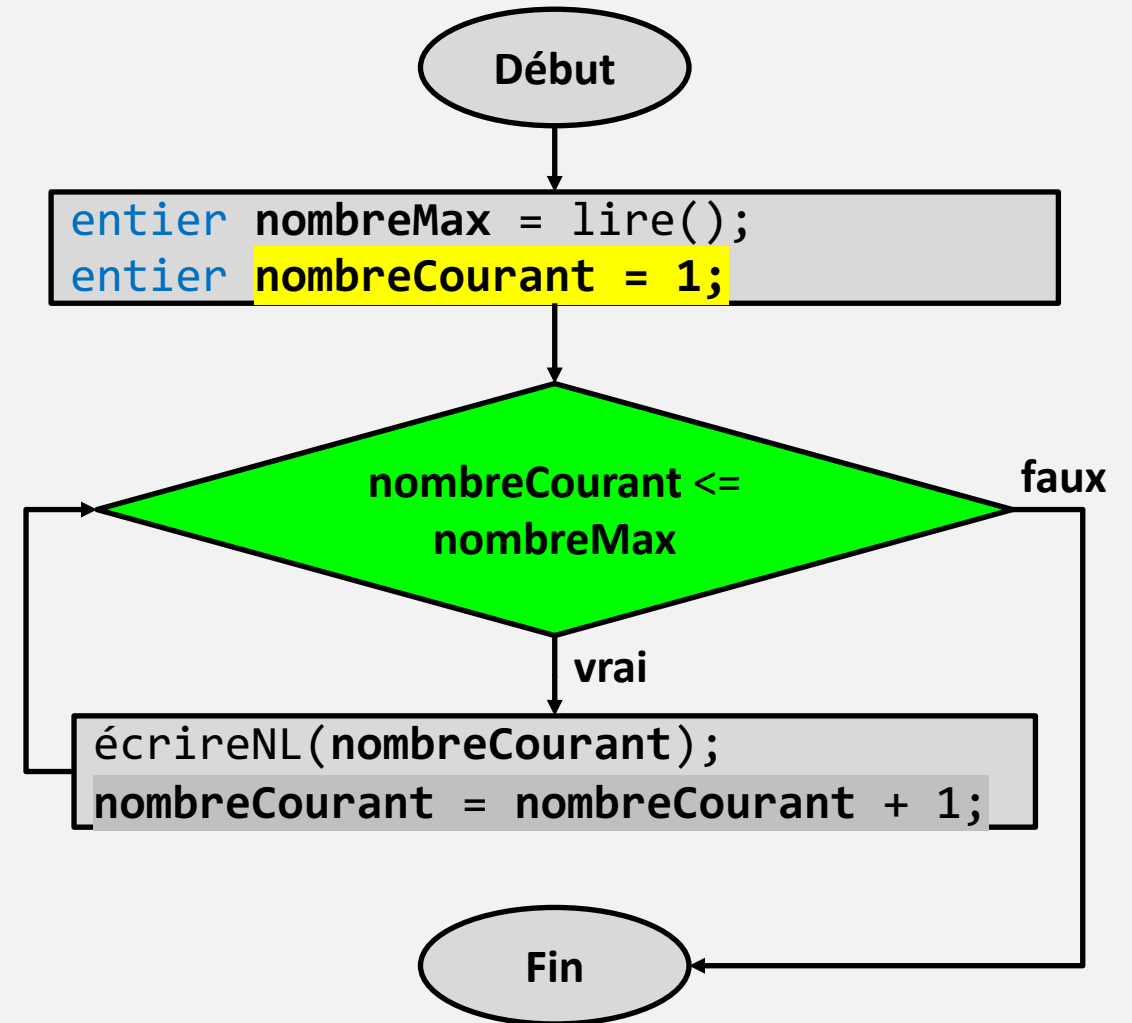
 : initialisation  
 : condition  
 : incrémentation

```
entier nombreMax = lire();  
pour entier nombreCourant de 1 à nombreMax {  
    écrireNL(nombreCourant);  
}
```




```
pour entier <nomVariable> de <valeurMin> à  
<valeurMax> [évolution (+|-)<incrément>] {  
    <algorithmeARépéter>  
}
```

La condition est évaluée avant de rentrer dans la boucle

Ici le nombre de répétitions est déterminé,  
soit égal à  $\text{<valeurMax>} - \text{<valeurMin>} + 1$



# Répétition « faire ... tant que »

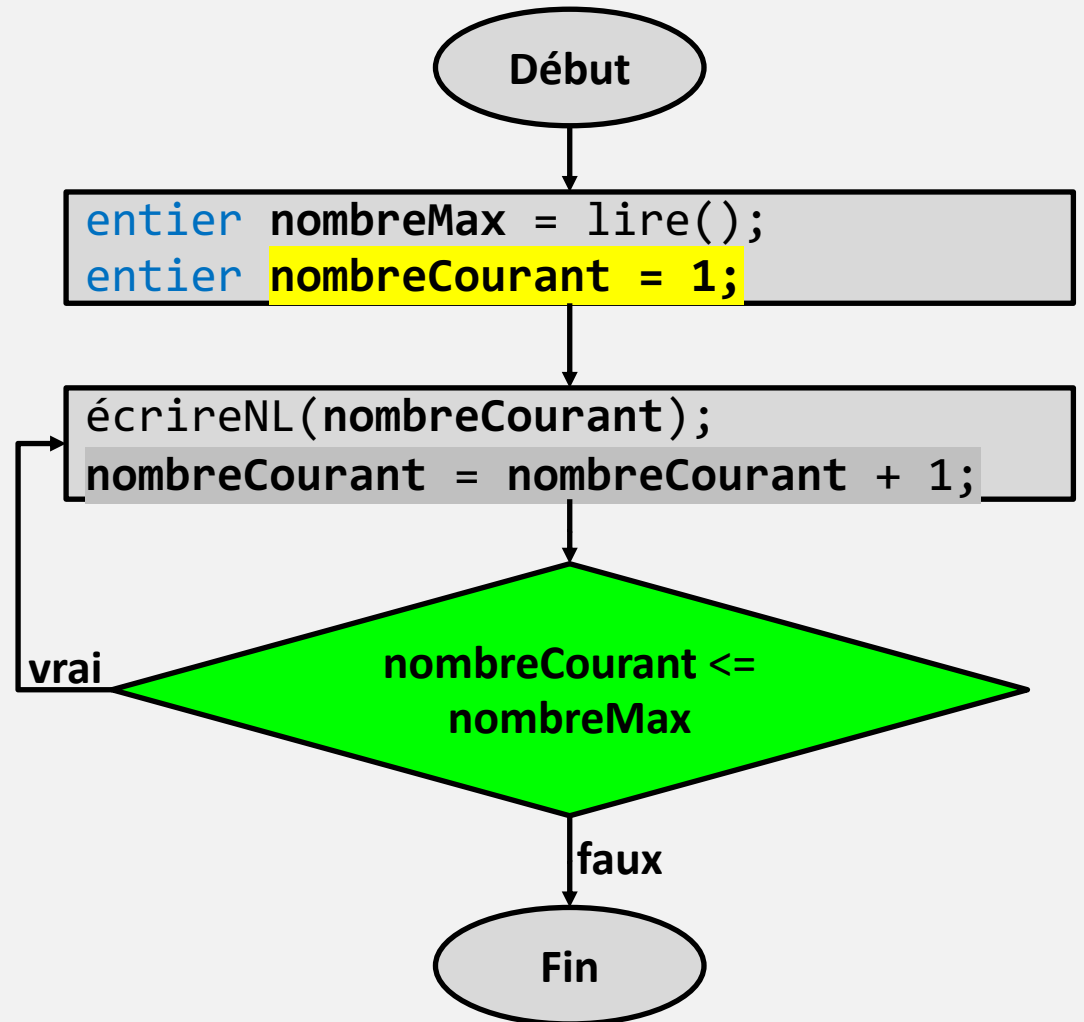
 : initialisation  
 : condition  
 : incrémentation

```
entier nombreMax = lire();  
entier nombreCourant = 1;  
faire {  
    écrireNL(nombreCourant);  
    nombreCourant = nombreCourant + 1;  
} tant que (nombreCourant <= nombreMax);
```

```
faire {  
    <algorithmeARépéter>  
} tant que (<conditionEstVraie>)
```

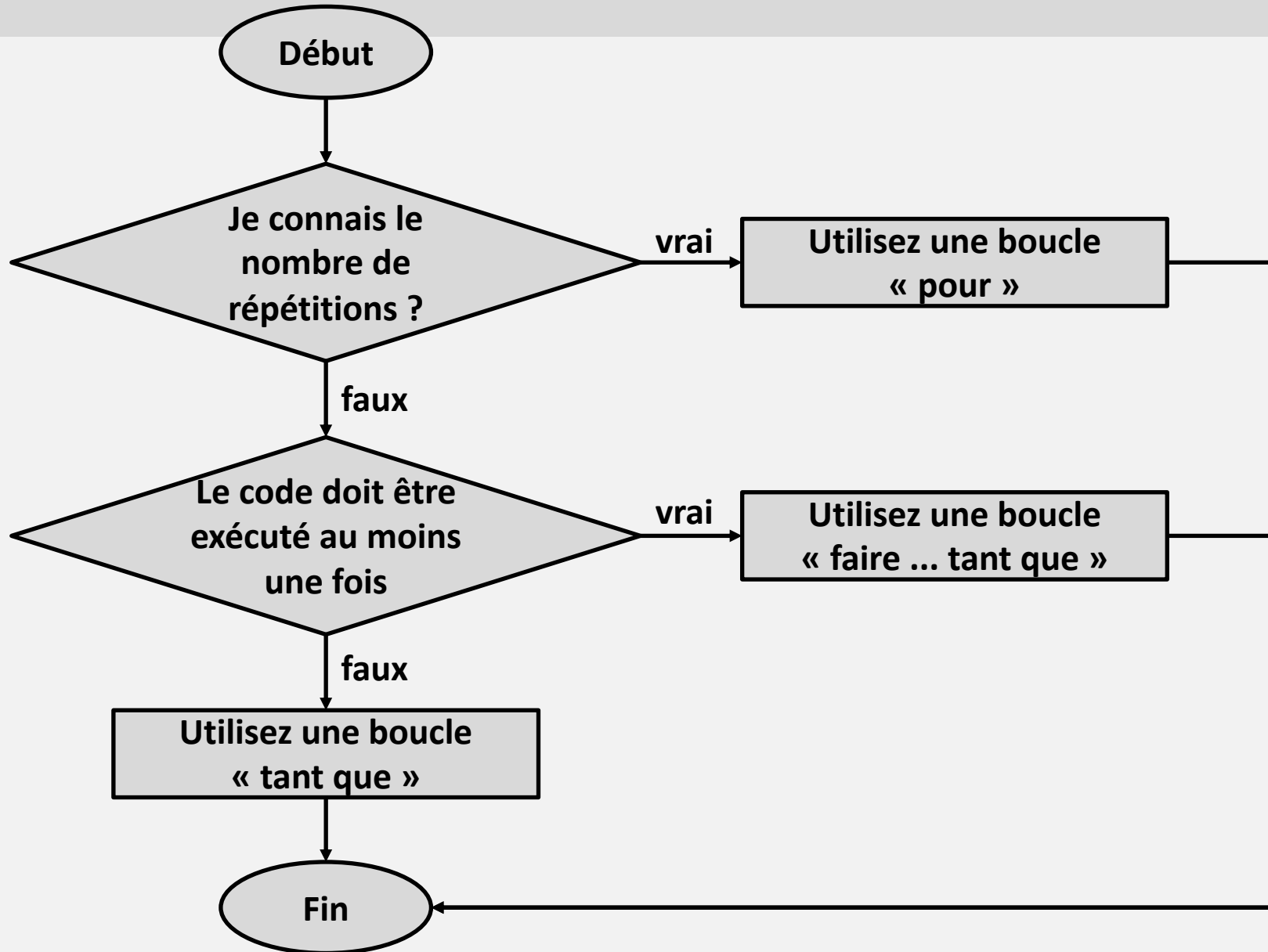
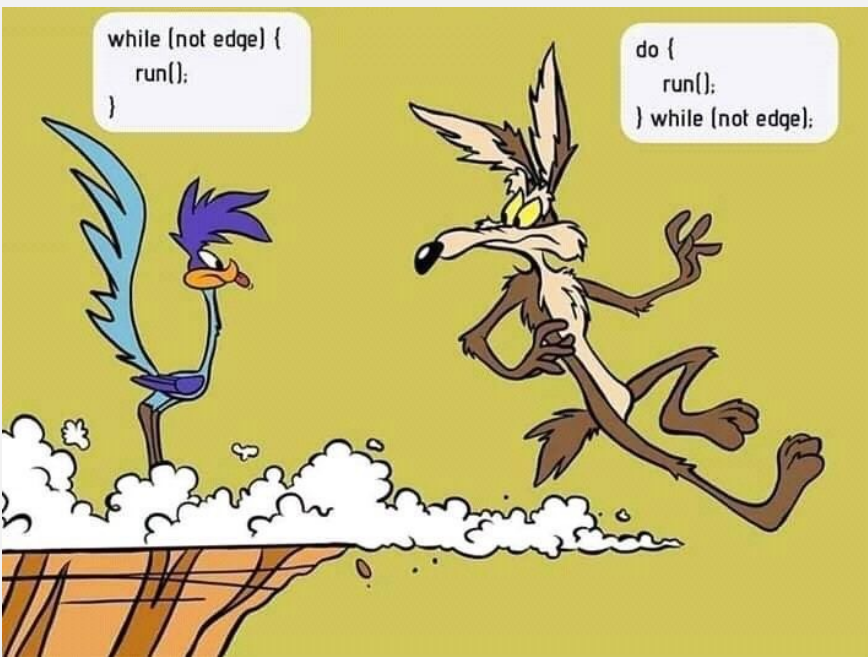
La condition est évaluée à la fin de la boucle

Ne pas oublier de faire évoluer ce qui est évalué comme condition  
sinon vous aurez une boucle infinie !





# Quand utiliser tel ou tel type de boucle ?



# Quand utiliser tel ou tel type de boucle ?

- faire ... tant que :
  - Saisie d'une valeur qui doit respecter une condition
  - On exécute au moins une fois les instructions à répéter
- tant que :
  - Tant que des valeurs du traitement ne sont pas arrivées à une valeur => on ne connaît pas le nombre d'itérations à l'avance
  - On ne rentre pas dans la boucle si la condition n'est pas vrai
- pour :
  - On « connaît » le nombre d'itérations
  - C'est la boucle la plus utilisée
  - Plus lisible la plupart du temps

# Et C# dans tout ça ? – tant que

	: initialisation
	: condition
	: incrémentation

```
entier nombreMax = lire();  
entier nombreCourant = 1;  
tant que (nombreCourant <= nombreMax) faire {  
    écrireNL(nombreCourant);  
    nombreCourant = nombreCourant + 1;  
}
```

```
int nombreMax = Console.In.ReadInt();  
int nombreCourant = 1;  
while (nombreCourant <= nombreMax) {  
    Console.Out.WriteLine(nombreCourant);  
    ++nombreCourant;  
}
```

# Et C# dans tout ça ? – pour

	: initialisation
	: condition
	: incrémentation

```
entier nombreMax = lire();  
pour entier nombreCourant de 1 à nombreMax {  
  
    écrireNL(nombreCourant);  
}
```

```
int nombreMax = Console.In.ReadInt();  
for (int nombreCourant = 1;  
    nombreCourant <= nombreMax;  
    nombreCourant = nombreCourant + 1) {  
    Console.Out.WriteLine(nombreCourant);  
}
```

# Et C# dans tout ca ? – faire ... tant que

	: initialisation
	: condition
	: incrémentation

```
entier nombreMax = lire();  
entier nombreCourant = 1;  
faire {  
    écrireNL(nombreCourant);  
    nombreCourant = nombreCourant + 1;  
} tant que (nombreCourant <= nombreMax);
```

```
int nombreMax = Console.In.ReadInt();  
int nombreCourant = 1;  
do {  
    Console.Out.WriteLine(nombreCourant);  
    ++nombreCourant;  
} while (nombreCourant <= nombreMax);
```

# Exemple C# complet

```
using System;

namespace Cours04Boucles
{
    class Program
    {
        static void Main(string[] args)
        {
            DemoBoucleFor();
            DemoBoucleWhile();
            DemoBoucleDoWhile();
        }

        public static void DemoBoucleFor()
        {
            int nombreMax = Console.In.ReadInt();
            for (int nombreCourant = 1;
                nombreCourant <= nombreMax;
                nombreCourant++)
            {
                Console.Out.WriteLine(nombreCourant);
            }
        }
    }
}
```

```
        public static void DemoBoucleWhile()
        {
            int nombreMax = Console.In.ReadInt();
            int nombreCourant = 1;
            while (nombreCourant <= nombreMax)
            {
                Console.Out.WriteLine(nombreCourant);
                ++nombreCourant;
            }
        }

        public static void DemoBoucleDoWhile()
        {
            int nombreMax = Console.In.ReadInt();
            int nombreCourant = 1;
            do
            {
                Console.Out.WriteLine(nombreCourant);
                ++nombreCourant;
            } while (nombreCourant <= nombreMax);
        }
    }
}
```