

Matière : Algorithmique et structure de données

Semestre : 01

Les Structures Itératives (Les Boucles)

M.M.DIAGNE

Plan

- 1 Introduction
- 2 La boucle «TantQue»
- 3 La boucle «Repeter-Jusqu a»
- 4 La boucle «Pour»
- 5 Les boucles Imbriquées

1. Introduction

1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.

1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

Algorithme Exemple1

Debut

Ecrire("2 puissance 0 = 1")

Ecrire("2 puissance 1 = 2")

Ecrire("2 puissance 2 = 4")

Fin

1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

Algorithme Exemple1

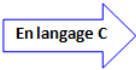
Debut

Ecrire("2 puissance 0 = 1")

Ecrire("2 puissance 1 = 2")

Ecrire("2 puissance 2 = 4")

Fin



En langage C

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("2 puissance 0 = 1");
```

```
    printf("2 puissance 1 = 2");
```

```
    printf("2 puissance 2 = 4");
```

```
    return 0;
```

```
}
```

1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

Algorithme Exemple1

Debut

Ecrire("2 puissance 0 = 1")

Ecrire("2 puissance 1 = 2")

Ecrire("2 puissance 2 = 4")

Fin



En langage C

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("2 puissance 0 = 1");
```

```
    printf("2 puissance 1 = 2");
```

```
    printf("2 puissance 2 = 4");
```

```
    return 0;
```

```
}
```

Question 1. Qu'est-ce-que fait l'algorithme (le programme) ?

1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

Algorithme Exemple1

Debut

Ecrire("2 puissance 0 = 1")

Ecrire("2 puissance 1 = 2")

Ecrire("2 puissance 2 = 4")

Fin

En langage C

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("2 puissance 0 = 1");
```

```
    printf("2 puissance 1 = 2");
```

```
    printf("2 puissance 2 = 4");
```

```
    return 0;
```

```
}
```

Question 1. Qu'est-ce-que fait l'algorithme (le programme) ?

Question 2. Écrire un algorithme qui affiche de 2^0 à 2^{64} ?

1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

Algorithme Exemple1

Debut

Ecrire("2 puissance 0 = 1")

Ecrire("2 puissance 1 = 2")

Ecrire("2 puissance 2 = 4")

Fin

En langage C

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("2 puissance 0 = 1");
```

```
    printf("2 puissance 1 = 2");
```

```
    printf("2 puissance 2 = 4");
```

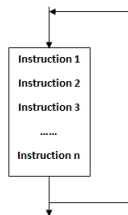
```
    return 0;
```

```
}
```

Question 1. Qu'est-ce-que fait l'algorithme (le programme) ?

Question 2. Écrire un algorithme qui affiche de 2^0 à 2^{64} ?

Solution : Structures itératives ou boucles.



1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

Algorithme Exemple1

Debut

Ecrire("2 puissance 0 = 1")

Ecrire("2 puissance 1 = 2")

Ecrire("2 puissance 2 = 4")

Fin

En langage C

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("2 puissance 0 = 1");
```

```
    printf("2 puissance 1 = 2");
```

```
    printf("2 puissance 2 = 4");
```

```
    return 0;
```

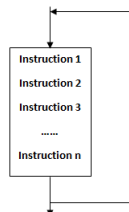
```
}
```

Question 1. Qu'est-ce-que fait l'algorithme (le programme) ?

Question 2. Écrire un algorithme qui affiche de 2^0 à 2^{64} ?

Solution : Structures itératives ou boucles.

1 TantQue...Faire ;



1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

Algorithme Exemple1

Debut

Ecrire("2 puissance 0 = 1")

Ecrire("2 puissance 1 = 2")

Ecrire("2 puissance 2 = 4")

Fin

En langage C

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
printf("2 puissance 0 = 1");
```

```
printf("2 puissance 1 = 2");
```

```
printf("2 puissance 2 = 4");
```

```
return 0;
```

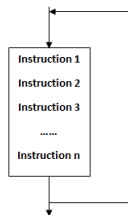
```
}
```

Question 1. Qu'est-ce-que fait l'algorithme (le programme) ?

Question 2. Écrire un algorithme qui affiche de 2^0 à 2^{64} ?

Solution : Structures itératives ou boucles.

- 1 TantQue...Faire ;
- 2 Repeter ... Jusqu a ;



1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

Algorithme Exemple1

Debut

Ecrire("2 puissance 0 = 1")

Ecrire("2 puissance 1 = 2")

Ecrire("2 puissance 2 = 4")

Fin

En langage C

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("2 puissance 0 = 1");
```

```
    printf("2 puissance 1 = 2");
```

```
    printf("2 puissance 2 = 4");
```

```
    return 0;
```

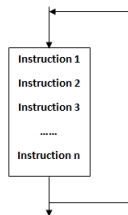
```
}
```

Question 1. Qu'est-ce-que fait l'algorithme (le programme) ?

Question 2. Écrire un algorithme qui affiche de 2^0 à 2^{64} ?

Solution : Structures itératives ou boucles.

- 1 TantQue...Faire ;
- 2 Repeter ... Jusqu a ;
- 3 Pour ... Faire.



1. Introduction

- Dans certaine situation, on est amené à répéter l'exécution d'une instruction (ou d'une liste d'instructions) plusieurs fois.
- Soit l'exemple suivant :

Algorithme Exemple1

Debut

Ecrire("2 puissance 0 = 1")

Ecrire("2 puissance 1 = 2")

Ecrire("2 puissance 2 = 4")

Fin

En langage C

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
printf("2 puissance 0 = 1");
```

```
printf("2 puissance 1 = 2");
```

```
printf("2 puissance 2 = 4");
```

```
return 0;
```

```
}
```

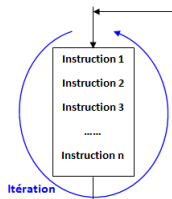
Question 1. Qu'est-ce-que fait l'algorithme (le programme) ?

Question 2. Écrire un algorithme qui affiche de 2^0 à 2^{64} ?

Solution : Structures itératives ou boucles.

- 1 TantQue...Faire ;
- 2 Repeter ... Jusqu a ;
- 3 Pour ... Faire.

- Le passage dans une boucle est appelé **itération**.



2. La boucle «TantQue ... Faire»

2. La boucle «TantQue ... Faire»

- Une instruction ou un groupe d'instructions est exécuté répétitivement tout le temps où une *condition* est *vraie*.

2. La boucle «TantQue ... Faire»

- Une instruction ou un groupe d'instructions est exécuté répétitivement tout le temps où une *condition* est *vraie*.

2. La boucle «TantQue ... Faire»

- Une instruction ou un groupe d'instructions est exécuté répétitivement tout le temps où une *condition* est *vraie*.

Syntaxe : Algorithmique

TantQue (**condition**) **Faire**
 //liste d'instructions

FinTantQue

2. La boucle «TantQue ... Faire»

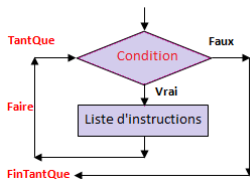
- Une instruction ou un groupe d'instructions est exécuté répétitivement tout le temps où une *condition* est *vraie*.

Syntaxe : Algorithmique

TantQue (**condition**) **Faire**
 //liste d'instructions

FinTantQue

Rep. Graph. (Organigramme)



Syntaxe : Langage C

```

while (condition)
{
    //liste d'instructions
}
  
```

Fonctionnement :

2. La boucle «TantQue ... Faire»

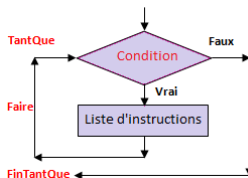
- Une instruction ou un groupe d'instructions est exécuté répétitivement tout le temps où une *condition* est *vraie*.

Syntaxe : Algorithmique

TantQue (**condition**) **Faire**
//liste d'instructions

FinTantQue

Rep. Graph. (Organigramme)



Syntaxe : Langage C

```
while (condition)
{
    //liste d'instructions
}
```

Fonctionnement :

- L'exécution de la boucle dépend de la valeur de la condition (**condition**).

2. La boucle «TantQue ... Faire»

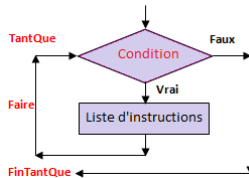
- Une instruction ou un groupe d'instructions est exécuté répétitivement tout le temps où une *condition* est *vraie*.

Syntaxe : Algorithmique

TantQue (**condition**) **Faire**
//liste d'instructions

FinTantQue

Rep. Graph. (Organigramme)



Syntaxe : Langage C

```

while (condition)
{
    //liste d'instructions
}

```

Fonctionnement :

- L'exécution de la boucle dépend de la valeur de la condition (**condition**).
- Si elle est **vraie**, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne **FinTantQue** .

2. La boucle «TantQue ... Faire»

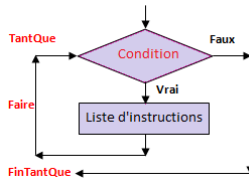
- Une instruction ou un groupe d'instructions est exécuté répétitivement tout le temps où une *condition* est *vraie*.

Syntaxe : Algorithmique

TantQue (**condition**) **Faire**
//liste d'instructions

FinTantQue

Rep. Graph. (Organigramme)



Syntaxe : Langage C

```
while (condition)
{
    //liste d'instructions
}
```

Fonctionnement :

- L'exécution de la boucle dépend de la valeur de la condition (**condition**).
- Si elle est **vraie**, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne **FinTantQue**.
- Il retourne ensuite sur la ligne du **TantQue**, procède au même examen, et ainsi de suite.

2. La boucle «TantQue ... Faire»

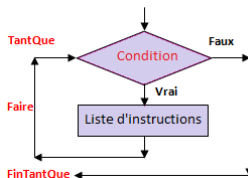
- Une instruction ou un groupe d'instructions est exécuté répétitivement tout le temps où une *condition* est *vraie*.

Syntaxe : Algorithmique

TantQue (**condition**) **Faire**
 //liste d'instructions

FinTantQue

Rep. Graph. (Organigramme)



Syntaxe : Langage C

```

while (condition)
{
    //liste d'instructions
}
  
```

Fonctionnement :

- L'exécution de la boucle dépend de la valeur de la condition (**condition**).
- Si elle est **vraie**, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne **FinTantQue**.
- Il retourne ensuite sur la ligne du **TantQue**, procède au même examen, et ainsi de suite.
- La boucle ne s'arrête que lorsque la condition (**condition**) prend la valeur **fausse**, dans ce cas le programme poursuit son exécution après **FinTantQue**.

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const $A = 2018$

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire ($A + N$)

Fin

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const A = 2018

Var N : Entier

P : Reel

Debut

N ← 0

P ← 42.2

TantQue (P ≤ 50) **Faire**

P ← P + 0.02 * P

N ← N + 1

FinTantQue

Ecrire (A + N)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P ≤ 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const A = 2018

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire (A + N)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P <= 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
-----------	---	---

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const A = 2018

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire (A + N)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P <= 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const A = 2018

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire (A + N)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P <= 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20
1	1	43.04

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const A = 2018

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire (A + N)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P <= 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20
1	1	43.04
2	2	43.90

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const A = 2018

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire (A + N)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P <= 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20
1	1	43.04
2	2	43.90
3	3	44.78

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const $A = 2018$

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire ($A + N$)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P <= 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20
1	1	43.04
2	2	43.90
3	3	44.78
4	4	45.68

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const A = 2018

Var N : Entier

P : Reel

Debut

N ← 0

P ← 42.2

TantQue (P ≤ 50) **Faire**

P ← P + 0.02 * P

N ← N + 1

FinTantQue

Ecrire (A + N)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P ≤ 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20
1	1	43.04
2	2	43.90
3	3	44.78
4	4	45.68
5	5	46.59

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const A = 2018

Var N : Entier

P : Reel

Debut

N ← 0

P ← 42.2

TantQue (P ≤ 50) **Faire**

P ← P + 0.02 * P

N ← N + 1

FinTantQue

Ecrire (A + N)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P ≤ 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20
1	1	43.04
2	2	43.90
3	3	44.78
4	4	45.68
5	5	46.59
6	6	47.52

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const $A = 2018$

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire ($A + N$)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P <= 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20
1	1	43.04
2	2	43.90
3	3	44.78
4	4	45.68
5	5	46.59
6	6	47.52
7	7	48.47

Exemple : TantQue...Faire

La population Algérienne à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const $A = 2018$

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire ($A + N$)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P <= 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20
1	1	43.04
2	2	43.90
3	3	44.78
4	4	45.68
5	5	46.59
6	6	47.52
7	7	48.47
8	8	49.44

Exemple : TantQue...Faire

La population SENEGALAISE à 42.2 millions d'habitants au 1 Janvier 2018. Elle augmente de 2% tous les ans (sous l'hypothèse).

Problème : Quand dépasse-t-on 50 millions ?

Solution : Algorithmique

Algorithme Population

Const A = 2018

Var N : Entier

P : Reel

Debut

$N \leftarrow 0$

$P \leftarrow 42.2$

TantQue ($P \leq 50$) **Faire**

$P \leftarrow P + 0.02 * P$

$N \leftarrow N + 1$

FinTantQue

Ecrire (A + N)

Fin

Solution : C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int A = 2018;
```

```
    int N = 0;
```

```
    float P = 42.2;
```

```
    while (P <= 50.0)
```

```
    {
```

```
        P = P + 0.02 * P;
```

```
        N = N + 1;
```

```
    }
```

```
    printf ("%d", A + N);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	N	P
0	0	42.20
1	1	43.04
2	2	43.90
3	3	44.78
4	4	45.68
5	5	46.59
6	6	47.52
7	7	48.47
8	8	49.44
9	9	50.43

3. La boucle «Repeter...Jusqu a»

3. La boucle «Repeter...Jusqu a»

- Cette boucle sert à répéter un traitement jusqu'à ce qu'une *condition* soit **vraie**.

3. La boucle «Repeter...Jusqu a»

- Cette boucle sert à répéter un traitement jusqu'à ce qu'une *condition* soit **vraie**.

Syntaxe : Algorithmique

Repeter

//liste d'instructions

Jusqu a (**condition**)

3. La boucle «Repeter...Jusqu a»

- Cette boucle sert à répéter un traitement jusqu'à ce qu'une *condition* soit **vraie**.

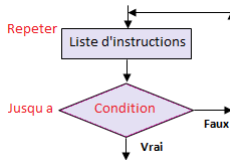
Syntaxe : Algorithmique

Repeter

```
//liste d'instructions
```

Jusqu a (condition)

Rep. Graph. (Organigramme)



3. La boucle «Repete...Jusqu a»

- Cette boucle sert à répéter un traitement jusqu'à ce qu'une *condition* soit **vraie**.

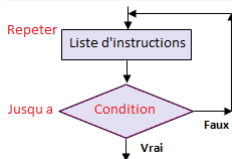
Syntaxe : Algorithmique

Repete

//liste d'instructions

Jusqu a (**condition**)

Rep. Graph. (Organigramme)



Syntaxe : Langage C

do

{

//liste d'instructions

} **while** (**condition**);

3. La boucle «Repete...Jusqu a»

- Cette boucle sert à répéter un traitement jusqu'à ce qu'une *condition* soit **vraie**.

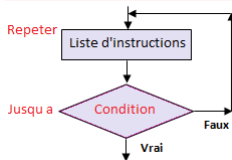
Syntaxe : Algorithmique

Repete

//liste d'instructions

Jusqu a (**condition**)

Rep. Graph. (Organigramme)



Syntaxe : Langage C

do

{

//liste d'instructions

} **while** (**condition**);

Fonctionnement :

3. La boucle «Repete...Jusqu a»

- Cette boucle sert à répéter un traitement jusqu'à ce qu'une *condition* soit **vraie**.

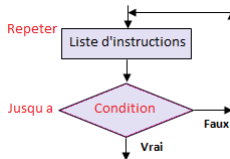
Syntaxe : Algorithmique

Repete

//liste d'instructions

Jusqu a (**condition**)

Rep. Graph. (Organigramme)



Syntaxe : Langage C

do

{

//liste d'instructions

} **while** (**condition**);

Fonctionnement :

3. La boucle «Repeter...Jusqu a»

- Cette boucle sert à répéter un traitement jusqu'à ce qu'une *condition* soit **vraie**.

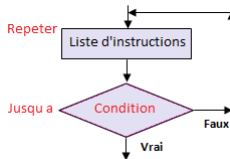
Syntaxe : Algorithmique

Repeter

//liste d'instructions

Jusqu a (**condition**)

Rep. Graph. (Organigramme)



Syntaxe : Langage C

do

{

//liste d'instructions

} **while** (**condition**);

Fonctionnement :

- La liste d'instructions est exécutée, puis la condition est évaluée ;

3. La boucle «Repeter...Jusqu a»

- Cette boucle sert à répéter un traitement jusqu'à ce qu'une *condition* soit **vraie**.

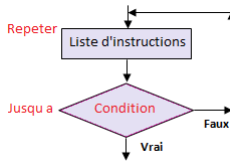
Syntaxe : Algorithmique

Repeter

```
//liste d'instructions
```

Jusqu a (condition)

Rep. Graph. (Organigramme)



Syntaxe : Langage C

do

 $\{$

```
//liste d'instructions
```

```
} while (condition);
```

Fonctionnement :

- La liste d'instructions est exécutée, puis la condition est évaluée ;
- si elle est **fausse**, le corps de la boucle est exécuté à nouveau puis la condition est réévaluée ;

3. La boucle «Repete...Jusqu a»

- Cette boucle sert à répéter un traitement jusqu'à ce qu'une *condition* soit **vraie**.

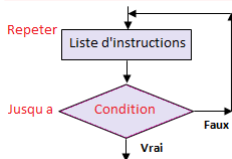
Syntaxe : Algorithmique

Repete

//liste d'instructions

Jusqu a (**condition**)

Rep. Graph. (Organigramme)



Syntaxe : Langage C

do

{

//liste d'instructions

} **while** (**condition**);

Fonctionnement :

- La liste d'instructions est exécutée, puis la condition est évaluée ;
- si elle est **fausse**, le corps de la boucle est exécuté à nouveau puis la condition est réévaluée ;
- si elle a la valeur **vrai**, le programme sort de la boucle et exécute l'instruction qui suit **Jusqu a**.

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication

Const $M = 9$

Var $i, n : \text{Entiers}$

Debut

Ecrire ("n ?")

Lire (n)

$i \leftarrow 0$

Repeter

$i \leftarrow i + 1$

Ecrire (n , "*", i ,

 "=", $i * n$)

Jusqu a ($i \geq M$)

Fin

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication

Const $M = 9$

Var $i, n : Entiers$

Debut

Ecrire ("n?")

Lire (n)

$i \leftarrow 0$

Repeter

$i \leftarrow i + 1$

Ecrire (n , "*", i ,

 "=", $i * n$)

Jusqu a ($i \geq M$)

Fin

Solution : Langage C

include <stdio.h>

int main()

{

const int $M = 9$;

int $i = 0, n = 0$;

do

 {

$i = i + 1$;

printf ("%d * %d =
 %d \n", $n, i, n * i$);

 } **while** ($i < M$);

return 0;

}

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication

Const $M = 9$

Var i, n : Entiers

Debut

Ecrire ("n?")

Lire (n)

$i \leftarrow 0$

Repeter

$i \leftarrow i + 1$

Ecrire (n , "*", i ,

 "=", $i * n$)

Jusqu a ($i \geq M$)

Fin

Solution : Langage C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int M = 9;
```

```
    int i = 0, n = 0;
```

```
    do
```

```
    {
```

```
        i = i + 1;
```

```
        printf ("%d * %d =\n", n, i, n * i);
```

```
    } while (i < M);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	Affichage
-----------	-----------

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication

Const $M = 9$

Var i, n : Entiers

Debut

Ecrire ("n?")

Lire (n)

$i \leftarrow 0$

Repeter

$i \leftarrow i + 1$

Ecrire (n , "*", i ,

 "=", $i * n$)

Jusqu a ($i \geq M$)

Fin

Solution : Langage C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int M = 9;
```

```
    int i = 0, n = 0;
```

```
    do
```

```
    {
```

```
        i = i + 1;
```

```
        printf ("%d * %d =\n", n, i, n * i);
```

```
    } while (i < M);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	Affichage
0	

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication
Const $M = 9$
Var i, n : Entiers
Debut
 Ecrire ("n?")
 Lire (n)
 $i \leftarrow 0$
 Repeter
 $i \leftarrow i + 1$
 Ecrire (n , "*", i ,
 "=", $i * n$)
 Jusqu a ($i \geq M$)
Fin

Solution : Langage C

```
# include <stdio.h>
int main()
{
    const int M = 9;
    int i = 0, n = 0;
    do
    {
        i = i + 1;
        printf ("%d * %d =
        %d \n ", n, i, n * i);
    } while (i < M);
    return 0;
}
```

Historique d'exécution

Itération	Affichage
0	
1	7*1=7

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication

Const $M = 9$

Var i, n : Entiers

Debut

Ecrire ("n?")

Lire (n)

$i \leftarrow 0$

Repeter

$i \leftarrow i + 1$

Ecrire (n , "*", i ,

 "=", $i * n$)

Jusqu a ($i \geq M$)

Fin

Solution : Langage C

include <stdio.h>

int main()

{

const int $M = 9$;

int $i = 0, n = 0$;

do

 {

$i = i + 1$;

printf ("%d * %d =

 %d \n ", $n, i, n * i$);

 } **while** ($i < M$);

return 0;

}

Historique d'exécution

Itération	Affichage
0	
1	7*1=7
2	7*2=14

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication
Const $M = 9$
Var i, n : Entiers
Debut
 Ecrire ("n?")
 Lire (n)
 $i \leftarrow 0$
 Repeter
 $i \leftarrow i + 1$
 Ecrire (n , "*", i ,
 "=", $i * n$)
 Jusqu a ($i \geq M$)
Fin

Solution : Langage C

```
# include <stdio.h>
int main()
{
    const int M = 9;
    int i = 0, n = 0;
    do
    {
        i = i + 1;
        printf ("%d * %d =
        %d \n ", n, i, n * i);
    } while (i < M);
    return 0;
}
```

Historique d'exécution

Itération	Affichage
0	
1	7*1=7
2	7*2=14
3	7*3=21

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication
Const $M = 9$
Var i, n : Entiers
Debut
 Ecrire ("n?")
 Lire (n)
 $i \leftarrow 0$
 Repeter
 $i \leftarrow i + 1$
 Ecrire (n , "*", i ,
 "=", $i * n$)
 Jusqu a ($i \geq M$)
Fin

Solution : Langage C

```
# include <stdio.h>
int main()
{
    const int M = 9;
    int i = 0, n = 0;
    do
    {
        i = i + 1;
        printf ("%d * %d =
        %d \n ", n, i, n * i);
    } while (i < M);
    return 0;
}
```

Historique d'exécution

Itération	Affichage
0	
1	7*1=7
2	7*2=14
3	7*3=21
4	7*4=28

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

```
Algorithme Multiplication
Const M = 9
Var i, n : Entiers
Debut
    Ecrire ("n?")
    Lire (n)
    i ← 0
    Repeter
        i ← i + 1
        Ecrire (n, "*", i,
            "=", i * n)
    Jusqu a (i >= M)
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    const int M = 9;
    int i = 0, n = 0;
    do
    {
        i = i + 1;
        printf ("%d * %d =
            %d \n ", n, i, n * i);
    } while (i < M);
    return 0;
}
```

Historique d'exécution

Itération	Affichage
0	
1	7*1=7
2	7*2=14
3	7*3=21
4	7*4=28
5	7*5=35

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication

Const $M = 9$

Var i, n : Entiers

Debut

Ecrire (" n ?")

Lire (n)

$i \leftarrow 0$

Repeter

$i \leftarrow i + 1$

Ecrire (n , "*", i ,

 "=", $i * n$)

Jusqu a ($i \geq M$)

Fin

Solution : Langage C

include <stdio.h>

int main()

{

const int $M = 9$;

int $i = 0, n = 0$;

do

 {

$i = i + 1$;

printf ("%d * %d =

 %d \n", $n, i, n * i$);

 } **while** ($i < M$);

return 0;

}

Historique d'exécution

Itération	Affichage
0	
1	7*1=7
2	7*2=14
3	7*3=21
4	7*4=28
5	7*5=35
6	7*6=42

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication

Const $M = 9$

Var i, n : Entiers

Debut

Ecrire (" n ?")

Lire (n)

$i \leftarrow 0$

Repeter

$i \leftarrow i + 1$

Ecrire (n , "*", i ,

 "=", $i * n$)

Jusqu a ($i \geq M$)

Fin

Solution : Langage C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    const int M = 9;
```

```
    int i = 0, n = 0;
```

```
    do
```

```
    {
```

```
        i = i + 1;
```

```
        printf ("%d * %d =
```

```
                %d \n ", n, i, n * i);
```

```
    } while (i < M);
```

```
    return 0;
```

```
}
```

Historique d'exécution

Itération	Affichage
0	
1	7*1=7
2	7*2=14
3	7*3=21
4	7*4=28
5	7*5=35
6	7*6=42
7	7*7=49

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication
Const $M = 9$
Var i, n : Entiers
Debut
 Ecrire (" n ?")
 Lire (n)
 $i \leftarrow 0$
 Repeter
 $i \leftarrow i + 1$
 Ecrire (n , "*", i ,
 "=", $i * n$)
 Jusqu a ($i \geq M$)
Fin

Solution : Langage C

```
# include <stdio.h>
int main()
{
    const int M = 9;
    int i = 0, n = 0;
    do
    {
        i = i + 1;
        printf ("%d * %d =
        %d \n ", n, i, n * i);
    } while (i < M);
    return 0;
}
```

Historique d'exécution

Itération	Affichage
0	
1	7*1=7
2	7*2=14
3	7*3=21
4	7*4=28
5	7*5=35
6	7*6=42
7	7*7=49
8	7*8=56

Exemple : Repeter...Jusqu a

Écrire un algorithme et la traduction en C d'un programme qui affiche la table de multiplication d'un nombre quelconque (saisi au clavier).

Solution : Algorithmique

Algorithme Multiplication
Const $M = 9$
Var i, n : Entiers
Debut
 Ecrire ("n?")
 Lire (n)
 $i \leftarrow 0$
 Repeter
 $i \leftarrow i + 1$
 Ecrire (n , "*", i ,
 "=", $i * n$)
 Jusqu a ($i \geq M$)
Fin

Solution : Langage C

```
# include <stdio.h>
int main()
{
    const int M = 9;
    int i = 0, n = 0;
    do
    {
        i = i + 1;
        printf ("%d * %d =
        %d \n ", n, i, n * i);
    } while (i < M);
    return 0;
}
```

Historique d'exécution

Itération	Affichage
0	
1	7*1=7
2	7*2=14
3	7*3=21
4	7*4=28
5	7*5=35
6	7*6=42
7	7*7=49
8	7*8=56
9	7*9=63

4. La boucle «Pour ... Faire»

4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

Syntaxe : Algorithmique

Pour *variable* **Allant de** *valeur-initiale* **A** *valeur-finale*

[pas] **Faire**

//liste d'instructions

FinPour

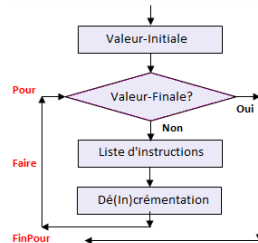
4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

Syntaxe : Algorithmique

Pour variable Allant de valeur-initiale **A** valeur-finale
 [pas] Faire
 //liste d'instructions
FinPour

Rep. Graph. (Organigramme)



4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

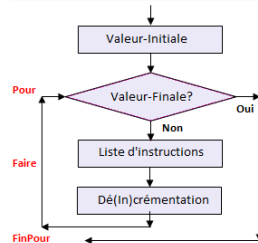
Syntaxe : Algorithmique

Pour variable Allant de valeur-initiale A valeur-finale
[pas] Faire
 //liste d'instructions
FinPour

Syntaxe : Langage C

```
for (initialisation ; condition ; pas)
{
    //liste d'instructions
}
```

Rep. Graph. (Organigramme)



4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

Syntaxe : Algorithmique

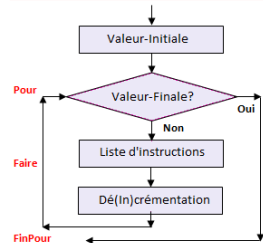
Pour variable Allant de valeur-initiale A valeur-finale
 [pas] Faire
 //liste d'instructions
FinPour

Syntaxe : Langage C

```
for (initialisation ; condition ; pas)
{
    //liste d'instructions
}
```

Fonctionnement :

Rep. Graph. (Organigramme)



4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

Syntaxe : Algorithmique

Pour *variable* **Allant de** *valeur-initiale* **A** *valeur-finale*
 [pas] **Faire**
 //liste d'instructions
FinPour

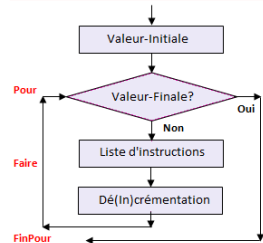
Syntaxe : Langage C

```
for (initialisation ; condition ; pas)
{
    //liste d'instructions
}
```

Fonctionnement :

- La *variable* compteur est de type entier. Elle est initialisée à la *valeur initiale* ;

Rep. Graph. (Organigramme)



4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

Syntaxe : Algorithmique

Pour variable Allant de valeur-initiale A valeur-finale
 [pas] Faire
 //liste d'instructions
FinPour

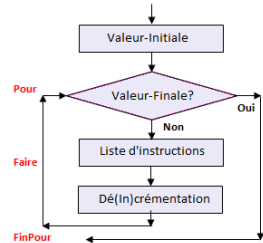
Syntaxe : Langage C

```
for (initialisation ; condition ; pas)
{
    //liste d'instructions
}
```

Fonctionnement :

- La **variable** compteur est de type entier. Elle est initialisée à la **valeur initiale** ;
- le compteur incrémente (ou décrémente) sa valeur de **pas** automatiquement à chaque tour de boucle jusqu'à la **valeur finale**.

Rep. Graph. (Organigramme)



4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

Syntaxe : Algorithmique

Pour *variable* **Allant de** *valeur-initiale* **A** *valeur-finale*
[pas] **Faire**
 //liste d'instructions
FinPour

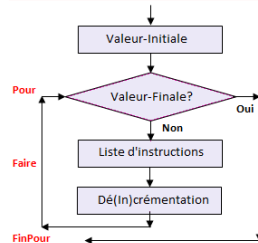
Syntaxe : Langage C

```
for (initialisation ; condition ; pas)
{
  //liste d'instructions
}
```

Fonctionnement :

- La *variable* compteur est de type entier. Elle est initialisée à la *valeur initiale* ;
- le compteur incrémente (ou décrémente) sa valeur de *pas* automatiquement à chaque tour de boucle jusqu'à la *valeur finale*.
- pour chaque valeur prise par la variable compteur, la liste des instructions est exécutée.

Rep. Graph. (Organigramme)



4. La boucle «Pour ... Faire»

- Cette boucle permet de répéter une liste d'instructions un nombre connu de fois.

Syntaxe : Algorithmique

Pour *variable* **Allant de** *valeur-initiale* **A** *valeur-finale*
 [*pas*] **Faire**
 //liste d'instructions
FinPour

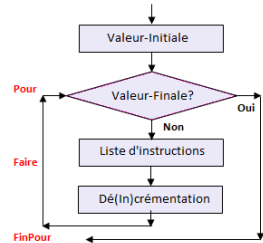
Syntaxe : Langage C

```
for (initialisation ; condition ; pas)
{
    //liste d'instructions
}
```

Fonctionnement :

- La *variable* compteur est de type entier. Elle est initialisée à la *valeur initiale* ;
- le compteur incrémente (ou décrémente) sa valeur de *pas* automatiquement à chaque tour de boucle jusqu'à la *valeur finale*.
- pour chaque valeur prise par la variable compteur, la liste des instructions est exécutée.
- lorsque la variable compteur vaut la *valeur-finale*, le traitement est exécuté une dernière fois puis l'algorithme sort de la boucle **Pour**.

Rep. Graph. (Organigramme)



Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

Algorithme Somme

Var S, i, N : Entiers

Debut

Ecrire ("Valeur de N ?")

Lire (N)

$S \leftarrow 0$;

Pour i **Allant de** 0 **A** N

Pas 2 **Faire**

$S \leftarrow S + i$

FinPour

Ecrire (S)

Fin

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

Algorithme Somme

Var S, i, N : Entiers

Debut

Ecrire ("Valeur de N ?")

Lire (N)

$S \leftarrow 0$;

Pour i **Allant de** 0 **A** N

Pas 2 **Faire**

$S \leftarrow S + i$

FinPour

Ecrire (S)

Fin

Solution : Langage C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    int  $S, i, N$ ;
```

```
    printf ("Valeur de  $N$  ?");
```

```
    scanf ("%d",& $N$ );
```

```
     $i = 0$ ;
```

```
     $S = 0$ ;
```

```
    for ( $i = 0$ ;  $i \leq N$ ;  $i + = 2$ )
```

```
    {
```

```
         $S + = i$ ;
```

```
    }
```

```
    printf ("%d",  $S$ );
```

```
    return 0;
```

```
}
```

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

Algorithme Somme

Var S, i, N : Entiers

Debut

Ecrire ("Valeur de N ?")

Lire (N)

$S \leftarrow 0$;

Pour i **Allant de** 0 **A** N

Pas 2 **Faire**

$S \leftarrow S + i$

FinPour

Ecrire (S)

Fin

Solution : Langage C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    int  $S, i, N$ ;
```

```
    printf ("Valeur de  $N$  ?");
```

```
    scanf ("%d",& $N$ );
```

```
     $i = 0$ ;
```

```
     $S = 0$ ;
```

```
    for ( $i = 0$ ;  $i \leq N$ ;  $i + = 2$ )
```

```
    {
```

```
         $S + = i$ ;
```

```
    }
```

```
    printf ("%d",  $S$ );
```

```
    return 0;
```

```
}
```

Historique d'exécution

It.	i	S
-----	---	---

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

```

Algorithme Somme
Var  $S, i, N$  : Entiers
Debut
    Ecrire ("Valeur de  $N$  ?")
    Lire ( $N$ )
     $S \leftarrow 0$ ;
    Pour  $i$  Allant de 0 A  $N$ 
Pas 2 Faire
         $S \leftarrow S + i$ 
    FinPour
    Ecrire ( $S$ )
Fin
  
```

Solution : Langage C

```

#include <stdio.h>
int main()
{
    int S, i, N;
    printf ("Valeur de N ?");
    scanf ("%d",&N);
    i = 0;
    S = 0;
    for (i = 0; i <= N; i += 2)
    {
        S += i;
    }
    printf ("%d", S);
    return 0;
}
  
```

Historique d'exécution

It.	i	S
0	?	0

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

```
Algorithme Somme
Var  $S, i, N$  : Entiers
Debut
    Ecrire ("Valeur de  $N$ ?")
    Lire ( $N$ )
     $S \leftarrow 0$ ;
    Pour  $i$  Allant de 0 A  $N$ 
Pas 2 Faire
     $S \leftarrow S + i$ 
FinPour
    Ecrire ( $S$ )
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $S, i, N$ ;
    printf ("Valeur de  $N$ ?");
    scanf ("%d",& $N$ );
     $i = 0$ ;
     $S = 0$ ;
    for ( $i = 0$ ;  $i \leq N$ ;  $i + = 2$ )
    {
         $S + = i$ ;
    }
    printf ("%d",  $S$ );
    return 0;
}
```

Historique d'exécution

It.	i	S
0	?	0
1	0	0

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

Algorithme Somme

Var S, i, N : Entiers

Debut

Ecrire ("Valeur de N ?")

Lire (N)

$S \leftarrow 0$;

Pour i **Allant de** 0 **A** N

Pas 2 **Faire**

$S \leftarrow S + i$

FinPour

Ecrire (S)

Fin

Solution : Langage C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    int  $S, i, N$ ;
```

```
    printf ("Valeur de  $N$  ?");
```

```
    scanf ("%d",& $N$ );
```

```
     $i = 0$ ;
```

```
     $S = 0$ ;
```

```
    for ( $i = 0$ ;  $i \leq N$ ;  $i + = 2$ )
```

```
    {
```

```
         $S + = i$ ;
```

```
    }
```

```
    printf ("%d",  $S$ );
```

```
    return 0;
```

```
}
```

Historique d'exécution

It.	i	S
0	?	0
1	0	0
2	2	2

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

Algorithme Somme

Var S, i, N : Entiers

Debut

Ecrire ("Valeur de N ?")

Lire (N)

$S \leftarrow 0$;

Pour i **Allant de** 0 **A** N

Pas 2 **Faire**

$S \leftarrow S + i$

FinPour

Ecrire (S)

Fin

Solution : Langage C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    int  $S, i, N$ ;
```

```
    printf ("Valeur de  $N$  ?");
```

```
    scanf ("%d",& $N$ );
```

```
     $i = 0$ ;
```

```
     $S = 0$ ;
```

```
    for ( $i = 0$ ;  $i \leq N$ ;  $i + = 2$ )
```

```
    {
```

```
         $S + = i$ ;
```

```
    }
```

```
    printf ("%d",  $S$ );
```

```
    return 0;
```

```
}
```

Historique d'exécution

It.	i	S
0	?	0
1	0	0
2	2	2
3	4	6

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

Algorithme Somme

Var S, i, N : Entiers

Debut

Ecrire ("Valeur de N ?")

Lire (N)

$S \leftarrow 0$;

Pour i **Allant de** 0 **A** N

Pas 2 **Faire**

$S \leftarrow S + i$

FinPour

Ecrire (S)

Fin

Solution : Langage C

```
# include <stdio.h>
```

```
int main()
```

```
{
```

```
    int  $S, i, N$ ;
```

```
    printf ("Valeur de  $N$  ?");
```

```
    scanf ("%d",&N);
```

```
     $i = 0$ ;
```

```
     $S = 0$ ;
```

```
    for ( $i = 0$ ;  $i \leq N$ ;  $i + = 2$ )
```

```
    {
```

```
         $S + = i$ ;
```

```
    }
```

```
    printf ("%d",  $S$ );
```

```
    return 0;
```

```
}
```

Historique d'exécution

It.	i	S
0	?	0
1	0	0
2	2	2
3	4	6
4	6	12

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

```
Algorithme Somme
Var  $S, i, N$  : Entiers
Debut
    Ecrire ("Valeur de  $N$ ?")
    Lire ( $N$ )
     $S \leftarrow 0$ ;
    Pour  $i$  Allant de 0 A  $N$ 
Pas 2 Faire
     $S \leftarrow S + i$ 
FinPour
    Ecrire ( $S$ )
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $S, i, N$ ;
    printf ("Valeur de  $N$ ?");
    scanf ("%d",&N);
     $i = 0$ ;
     $S = 0$ ;
    for ( $i = 0$ ;  $i \leq N$ ;  $i + = 2$ )
    {
         $S + = i$ ;
    }
    printf ("%d",  $S$ );
    return 0;
}
```

Historique d'exécution

It.	i	S
0	?	0
1	0	0
2	2	2
3	4	6
4	6	12
5	8	20

Exemple : Pour...Faire

Écrire un algorithme et la traduction en C d'un programme qui permet de saisir un nombre entier N et qui calcule la somme des entiers pairs jusqu'à ce nombre.

Solution : Algorithmique

```
Algorithme Somme
Var  $S, i, N$  : Entiers
Debut
    Ecrire ("Valeur de  $N$ ?")
    Lire ( $N$ )
     $S \leftarrow 0$ ;
    Pour  $i$  Allant de 0 A  $N$ 
Pas 2 Faire
     $S \leftarrow S + i$ 
FinPour
    Ecrire ( $S$ )
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $S, i, N$ ;
    printf ("Valeur de  $N$ ?");
    scanf ("%d",&N);
     $i = 0$ ;
     $S = 0$ ;
    for ( $i = 0$ ;  $i \leq N$ ;  $i + = 2$ )
    {
         $S + = i$ ;
    }
    printf ("%d",  $S$ );
    return 0;
}
```

Historique d'exécution

It.	i	S
0	?	0
1	0	0
2	2	2
3	4	6
4	6	12
5	8	20
6	8	30

5. Les boucles imbriquées

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Exemple : $N = 3$.

```
*  
**  
***
```

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Exemple : $N = 3$.

*
**

Solution : Algorithmique

Algorithme Triangle

Var N, i, j : Entiers

Debut

Ecrire ("Valeur de N ?")

Lire (N)

Pour i **Allant de** 1 **A** N

Faire

Pour j **Allant de** 1 **A** i

Faire

Ecrire ("*")

FinPour

FinPour

Fin

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Exemple : $N = 3$.

```
*  
**  
***
```

Solution : Algorithmique

```
Algorithme Triangle  
Var  $N, i, j$  : Entiers  
Debut  
    Ecrire ("Valeur de  $N$  ?")  
    Lire ( $N$ )  
    Pour  $i$  Allant de 1 A  $N$   
    Faire  
        Pour  $j$  Allant de 1 A  $i$   
        Faire  
            Ecrire ("*")  
        FinPour  
    FinPour  
Fin
```

Solution : Langage C

```
# include <stdio.h>  
int main()  
{  
    int  $N$ ;  
    printf ("Valeur de  $N$  ?");  
    scanf ("%d",& $N$ );  
    for (int  $i$  = 1 ;  $i$  <=  $N$  ;  $i$ ++)  
    {  
        for (int  $j$  = 1 ;  $j$  <=  $i$  ;  $j$ ++)  
        {  
            printf ("*");  
        }  
        printf ("\n");  
    }  
    return 0;  
}
```

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Solution : Algorithmique

```
Algorithme Triangle
Var  $N, i, j$  : Entiers
Debut
    Ecrire ("Valeur de  $N$  ?")
    Lire ( $N$ )
    Pour  $i$  Allant de 1 A  $N$ 
Faire
    Pour  $j$  Allant de 1 A  $i$ 
Faire
        Ecrire ("*")
    FinPour
FinPour
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $N$ ;
    printf ("Valeur de  $N$  ?");
    scanf ("%d",& $N$ );
    for (int  $i$  = 1;  $i$  <=  $N$ ;  $i$ ++)
    {
        for (int  $j$  = 1;  $j$  <=  $i$ ;  $j$ ++)
        {
            printf ("*");
        }
        printf ("\n");
    }
    return 0;
}
```

Exemple : $N = 3$.

```
*
**
***
```

Historique d'exécution

lt.	i	j

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Solution : Algorithmique

```
Algorithme Triangle
Var  $N, i, j$  : Entiers
Debut
    Ecrire ("Valeur de  $N$  ?")
    Lire ( $N$ )
    Pour  $i$  Allant de 1 A  $N$ 
Faire
    Pour  $j$  Allant de 1 A  $i$ 
Faire
        Ecrire ("*")
    FinPour
FinPour
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $N$ ;
    printf ("Valeur de  $N$  ?");
    scanf ("%d",& $N$ );
    for (int  $i$  = 1;  $i$  <=  $N$ ;  $i$ ++)
    {
        for (int  $j$  = 1;  $j$  <=  $i$ ;  $j$ ++)
        {
            printf ("*");
        }
        printf ("\n");
    }
    return 0;
}
```

Exemple : $N = 3$.

```
*
**
***
```

Historique d'exécution

lt.	i	j
0		

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Solution : Algorithmique

```
Algorithme Triangle
Var  $N, i, j$  : Entiers
Debut
    Ecrire ("Valeur de  $N$  ?")
    Lire ( $N$ )
    Pour  $i$  Allant de 1 A  $N$ 
Faire
    Pour  $j$  Allant de 1 A  $i$ 
Faire
        Ecrire ("*")
    FinPour
FinPour
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $N$ ;
    printf ("Valeur de  $N$  ?");
    scanf ("%d",& $N$ );
    for (int  $i$  = 1;  $i$  <=  $N$ ;  $i$ ++)
    {
        for (int  $j$  = 1;  $j$  <=  $i$ ;  $j$ ++)
        {
            printf ("*");
        }
        printf ("\n");
    }
    return 0;
}
```

Exemple : $N = 3$.

```
*
**
***
```

Historique d'exécution

It.	i	j
0		
1	1	1

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Solution : Algorithmique

```
Algorithme Triangle
Var  $N, i, j$  : Entiers
Debut
    Ecrire ("Valeur de  $N$  ?")
    Lire ( $N$ )
    Pour  $i$  Allant de 1 A  $N$ 
Faire
    Pour  $j$  Allant de 1 A  $i$ 
Faire
        Ecrire ("*")
    FinPour
FinPour
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $N$ ;
    printf ("Valeur de  $N$  ?");
    scanf ("%d",& $N$ );
    for (int  $i$  = 1;  $i$  <=  $N$ ;  $i$ ++)
    {
        for (int  $j$  = 1;  $j$  <=  $i$ ;  $j$ ++)
        {
            printf ("*");
        }
        printf ("\n");
    }
    return 0;
}
```

Exemple : $N = 3$.

```
*
**
***
```

Historique d'exécution

It.	i	j
0		
1	1	1
2	2	1

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Solution : Algorithmique

```
Algorithme Triangle
Var  $N, i, j$  : Entiers
Debut
    Ecrire ("Valeur de  $N$  ?")
    Lire ( $N$ )
    Pour  $i$  Allant de 1 A  $N$ 
Faire
    Pour  $j$  Allant de 1 A  $i$ 
Faire
        Ecrire ("*")
    FinPour
FinPour
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $N$ ;
    printf ("Valeur de  $N$  ?");
    scanf ("%d",& $N$ );
    for (int  $i$  = 1;  $i$  <=  $N$ ;  $i$ ++)
    {
        for (int  $j$  = 1;  $j$  <=  $i$ ;  $j$ ++)
        {
            printf ("*");
        }
        printf ("\n");
    }
    return 0;
}
```

Exemple : $N = 3$.

```
*
**
***
```

Historique d'exécution

It.	i	j
0		
1	1	1
2	2	1
3	2	2

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Solution : Algorithmique

```
Algorithme Triangle
Var  $N, i, j$  : Entiers
Debut
    Ecrire ("Valeur de  $N$  ?")
    Lire ( $N$ )
    Pour  $i$  Allant de 1 A  $N$ 
Faire
    Pour  $j$  Allant de 1 A  $i$ 
Faire
        Ecrire ("*")
    FinPour
FinPour
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $N$ ;
    printf ("Valeur de  $N$  ?");
    scanf ("%d",& $N$ );
    for (int  $i$  = 1;  $i$  <=  $N$ ;  $i$ ++)
    {
        for (int  $j$  = 1;  $j$  <=  $i$ ;  $j$ ++)
        {
            printf ("*");
        }
        printf ("\n");
    }
    return 0;
}
```

Exemple : $N = 3$.

```
*
**
***
```

Historique d'exécution

It.	i	j
0		
1	1	1
2	2	1
3	2	2
4	3	1

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Solution : Algorithmique

```
Algorithme Triangle
Var  $N, i, j$  : Entiers
Debut
    Ecrire ("Valeur de  $N$  ?")
    Lire ( $N$ )
    Pour  $i$  Allant de 1 A  $N$ 
Faire
    Pour  $j$  Allant de 1 A  $i$ 
Faire
        Ecrire ("*")
    FinPour
FinPour
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $N$ ;
    printf ("Valeur de  $N$  ?");
    scanf ("%d",& $N$ );
    for (int  $i$  = 1;  $i$  <=  $N$ ;  $i$ ++)
    {
        for (int  $j$  = 1;  $j$  <=  $i$ ;  $j$ ++)
        {
            printf ("*");
        }
        printf ("\n");
    }
    return 0;
}
```

Exemple : $N = 3$.

```
*
**
***
```

Historique d'exécution

It.	i	j
0		
1	1	1
2	2	1
3	2	2
4	3	1
5	3	2

5. Les boucles imbriquées

- La liste d'instructions d'une boucle peut contenir elle même une autre boucle.
C'est ce qu'on appelle des **boucles imbriquées**

Exercice

Écrire un algorithme (programme) qui demande à l'utilisateur un entier N puis dessine un triangle rectangle d'étoiles.

Solution : Algorithmique

```
Algorithme Triangle
Var  $N, i, j$  : Entiers
Debut
    Ecrire ("Valeur de  $N$  ?")
    Lire ( $N$ )
    Pour  $i$  Allant de 1 A  $N$ 
Faire
    Pour  $j$  Allant de 1 A  $i$ 
Faire
        Ecrire ("*")
    FinPour
FinPour
Fin
```

Solution : Langage C

```
# include <stdio.h>
int main()
{
    int  $N$ ;
    printf ("Valeur de  $N$  ?");
    scanf ("%d",& $N$ );
    for (int  $i$  = 1;  $i$  <=  $N$ ;  $i$ ++)
    {
        for (int  $j$  = 1;  $j$  <=  $i$ ;  $j$ ++)
        {
            printf ("*");
        }
        printf ("\n");
    }
    return 0;
}
```

Exemple : $N = 3$.

```
*
**
***
```

Historique d'exécution

It.	i	j
0		
1	1	1
2	2	1
3	2	2
4	3	1
5	3	2
6	3	3

Quelle boucle puisse-je utiliser pour mon programme C ?

Quelle boucle puisse-je utiliser pour mon programme C ?

Utilisez la structure qui reflète le mieux l'idée du programme que vous voulez réaliser :

Quelle boucle puisse-je utiliser pour mon programme C ?

Utilisez la structure qui reflète le mieux l'idée du programme que vous voulez réaliser :

- Si la liste d'instructions ne doit pas être exécutée si la condition est **fausse**, alors utilisez **for** ou **while**.

Quelle boucle puisse-je utiliser pour mon programme C ?

Utilisez la structure qui reflète le mieux l'idée du programme que vous voulez réaliser :

- Si la liste d'instructions ne doit pas être exécutée si la condition est **fausse**, alors utilisez **for** ou **while**.
- Si la liste d'instructions doit être exécutée au moins une fois, alors utilisez **do-while** .

Quelle boucle puisse-je utiliser pour mon programme C ?

Utilisez la structure qui reflète le mieux l'idée du programme que vous voulez réaliser :

- Si la liste d'instructions ne doit pas être exécutée si la condition est **fausse**, alors utilisez **for** ou **while**.
- Si la liste d'instructions doit être exécutée au moins une fois, alors utilisez **do-while** .
- Si le nombre d'exécutions de la liste d'instructions est connu à l'avance alors utilisez **for**.

Quelle boucle puisse-je utiliser pour mon programme C ?

Utilisez la structure qui reflète le mieux l'idée du programme que vous voulez réaliser :

- Si la liste d'instructions ne doit pas être exécutée si la condition est **fausse**, alors utilisez **for** ou **while**.
- Si la liste d'instructions doit être exécutée au moins une fois, alors utilisez **do-while** .
- Si le nombre d'exécutions de la liste d'instructions est connu à l'avance alors utilisez **for**.
- Si la liste d'instructions doit être exécutée aussi longtemps qu'une condition extérieure est **vraie** alors utilisez **while**.

Quelle boucle puisse-je utiliser pour mon programme C ?

Utilisez la structure qui reflète le mieux l'idée du programme que vous voulez réaliser :

- Si la liste d'instructions ne doit pas être exécutée si la condition est **fausse**, alors utilisez **for** ou **while**.
- Si la liste d'instructions doit être exécutée au moins une fois, alors utilisez **do-while** .
- Si le nombre d'exécutions de la liste d'instructions est connu à l'avance alors utilisez **for**.
- Si la liste d'instructions doit être exécutée aussi longtemps qu'une condition extérieure est **vraie** alors utilisez **while**.

Le choix entre **while** et **for** n'est souvent qu'une question de préférence ou d'habitudes.