

- Comment calculer la factorielle  $n!$  d'un entier  $n \geq 0$  ?

Par une **itération** (comme à la main !) :

$$n! = 1 \times 2 \times \dots \times i \times (i+1) \times \dots \times n$$

*Par exemple  
avec une  
boucle **for***

```
int fac(int n) {
  int res = 1;
  for (int i = 2; i <= n; i = i+1) {
    res = res * i;
  }
  return res;
}
```

*déclaration et  
initialisation du résultat*

*calcul du résultat*

*retour du résultat*

- **Méthode itérative**  $\equiv$  programmée avec une boucle.

• Traduction en français et exécution du calcul de  $\text{fac}(5)$  :

*Methode*  $\text{fac} : \mathbb{N} \rightarrow \mathbb{N}$

• *Soit*  $\text{res}$  *un entier initialisé à 1*

• *Pour chaque*  $i$  *depuis 2*

*tant que*  $i \leq n$

*puis chaque fois*  $i$  *est augmenté de 1*

*multiplier*  $\text{res}$  *par*  $i$

• *Le résultat est*  $\text{res}$

• On dit que les variables  $\text{res}$  et  $i$  qui varient durant l'exécution de la boucle `for`, sont des **VARIABLES DE BOUCLE**. La variable  $n$  ne varie pas, ce  $n$  est pas une variable de boucle.

`fac(5);` // ==120

```


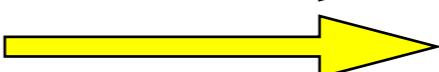
int fac(int n) {
  int res = 1;
  for (int i = 2; i <= n; i = i + 1) {
    res = res * i;
  }
  return res;
}
  
```

- *En principe*, dans une boucle `for`, le nombre de tours de boucle est connu à l'avance.
- Lorsqu'on ne sait pas combien de fois on va *boucler*, il faut opter pour l'une des boucles **while...** ou **do...while...**
- Exemple : soit à calculer le **plus petit diviseur**  $d \geq 2$  d'un entier  $n \geq 2$ . Autrement dit, son plus petit facteur premier.

```
int ppdiv(int n) {
    int d = 2;
    while (n % d != 0) {
        d = d + 1;
    }
    return d;
}
```

*Je pars du premier candidat  $d = 2$ , et tant que  $d$  ne divise pas  $n$ , je monte !*

*Dans le pire des cas, j'obtiendrai  $d = n$*

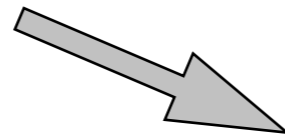
<code>ppdiv(2003)</code>		2003
<code>ppdiv(2009)</code>		7

- Reprenons l'exemple de la factorielle.

Traduction du for en while :

```
int fac(int n) {           // n >= 0
    int res = 1;
    for (int i = 1; i <= n; i = i+1) {
        res = res * i;
    }
    return res;
}
```

*La variable i est  
locale à la boucle*



```
int fac(int n) {
    int res = 1, i = 1;
    while (i <= n) {
        res = res * i;
        i = i + 1;
    }
    return res;
}
```

*La variable i existe  
en-dehors de la boucle*

```
for(A ; B ; C) {
    D;
}
```

≡

```
A;
while (B) {
    D;
    C;
}
```

- Analogue à la boucle `while`, elle échange l'ordre du test et des instructions. Les instructions sont effectuées avant le test, donc **au moins une fois !**
- Reprenons le calcul du **plus petit diviseur**  $d \geq 2$  d'un entier  $n \geq 2$ .

```

int ppdiv(int n) { // n >= 2
  int d = 1;
  do {
    d = d + 1;
  } while (n % d != 0);
  return d;
}
  
```

<pre>do {   A; } while(B);</pre>	$\equiv$	<pre>A; while (B) {   A; }</pre>
----------------------------------	----------	----------------------------------

# Méthodologie de construction d'une boucle

- Avec le principe de récurrence (simple), on supposait avoir fait 99% du calcul, et on faisait le dernier pas : passage de  $n-1$  à  $n$ .
- Avec une boucle, c'est plus compliqué. En contrepartie, on peut décrire l'état du calcul en plein milieu de son exécution.
  - Commencez par donner un nom au résultat s'il y en a, et n'oubliez pas de l'initialiser.
  - Imaginez-vous en plein milieu du calcul. Quelle est la situation ? De quelles variables avez-vous besoin pour décrire ce que la boucle a déjà fait ou calculé ? Quelle relation y a-t-il entre ces variables ?
- L'état du calcul en plein milieu d'une boucle est donné par l'ensemble des valeurs  $x_i$  des variables  $v_i$  de boucle. On dit que  $((x_1, v_1), \dots, (x_n, v_n))$  est le **vecteur d'état** de la boucle.

## ***EXEMPLE DETAILLE** : factorisation d'un entier*

- Soit à faire afficher la suite de tous les diviseurs premiers d'un entier  $n \geq 2$ . Par exemple, pour  $n = 1144660$ , on veut voir :

Décomposition de 1144660 : 2 2 5 11 11 11 43

- Je me place *en plein milieu du calcul*. Quelle est la *situation* ? Mon entier  $n$  a déjà été purgé d'un certain nombre de facteurs premiers qui ont été affichés, et j'en suis au candidat  $p \geq 2$  que je n'ai pas encore traité.



*Méthode afficheDiviseursPremiers :  $\mathbb{N} \rightarrow \emptyset$*

- *On suppose que  $n \geq 2$*
- *Soit  $p$  le candidat diviseur courant initialisé à 2*
- *Tant que  $n > 1$ , donc tant qu'il reste des facteurs :*  
*Si le candidat  $p$  divise  $n$  : afficher  $p$  et diviser  $n$  par  $p$*   
*sinon faire avancer  $p$*
- *Aucun résultat*

- Il ne reste qu'à passer de ce **pseudo-langage** (intermédiaire entre le bon français et notre langage de programmation) à un codage propre :

```

void afficheDiviseursPremiers(int n) {
    if (n < 2) return;           // si n < 2, je ne joue pas !
    int candidat = 2;           // donc n ≥ 2 et candidat = 2 est le premier candidat
    while (n > 1) {             // tant qu'il reste des diviseurs non purgés
        if (n % candidat == 0) { // si candidat divise n
            print(candidat + " "); // on affiche le diviseur qu'on a trouvé
            n = n / candidat;      // puis on purge candidat de n, et n diminue
        } else {                  // sinon
            candidat = candidat + 1; // on passe au candidat suivant
        }
    }
    println();                    // pour aller à la ligne suivante !
}

```

- Au fur et à mesure que p monte, l'entier n diminue et finit par converger vers 1. Ce phénomène de **vases communicants** se rencontre assez fréquemment dans les boucles.

*N.B. La boucle ci-dessus aurait du être un do..while... Pourquoi ?*



- Il ne reste qu'à passer de ce **pseudo-langage** (intermédiaire entre le bon français et notre langage de programmation) à un codage propre :

```

void afficheDiviseursPremiers(int n) {
    if (n < 2) return;           // si n < 2, je ne joue pas !
    int candidat = 2;           // donc n ≥ 2 et candidat = 2 est le premier candidat
    do {                         // tant qu'il reste des diviseurs non purgés
        if (n % candidat == 0) { // si candidat divise n
            print(candidat + " "); // on affiche le diviseur qu'on a trouvé
            n = n / candidat;      // puis on purge candidat de n, et n diminue
        } else {                 // sinon
            candidat = candidat + 1; // on passe au candidat suivant
        }
    } while (n > 1);
    println();                   // pour aller à la ligne suivante !
}
  
```

- Au fur et à mesure que p monte, l'entier n diminue et finit par converger vers 1. Ce phénomène de **vases communicants** se rencontre assez fréquemment dans les boucles.

- Gourmandise : je souhaite en plus l'affichage avec des exposants :

Décomposition de 1144660 : 2(2) 5(1) 11(3) 43(1)

- Rebelotte. Je suis en plein milieu du calcul, le candidat courant est p et j'ai déjà compté k fois le facteur p. Nouvelle variable de boucle !

```

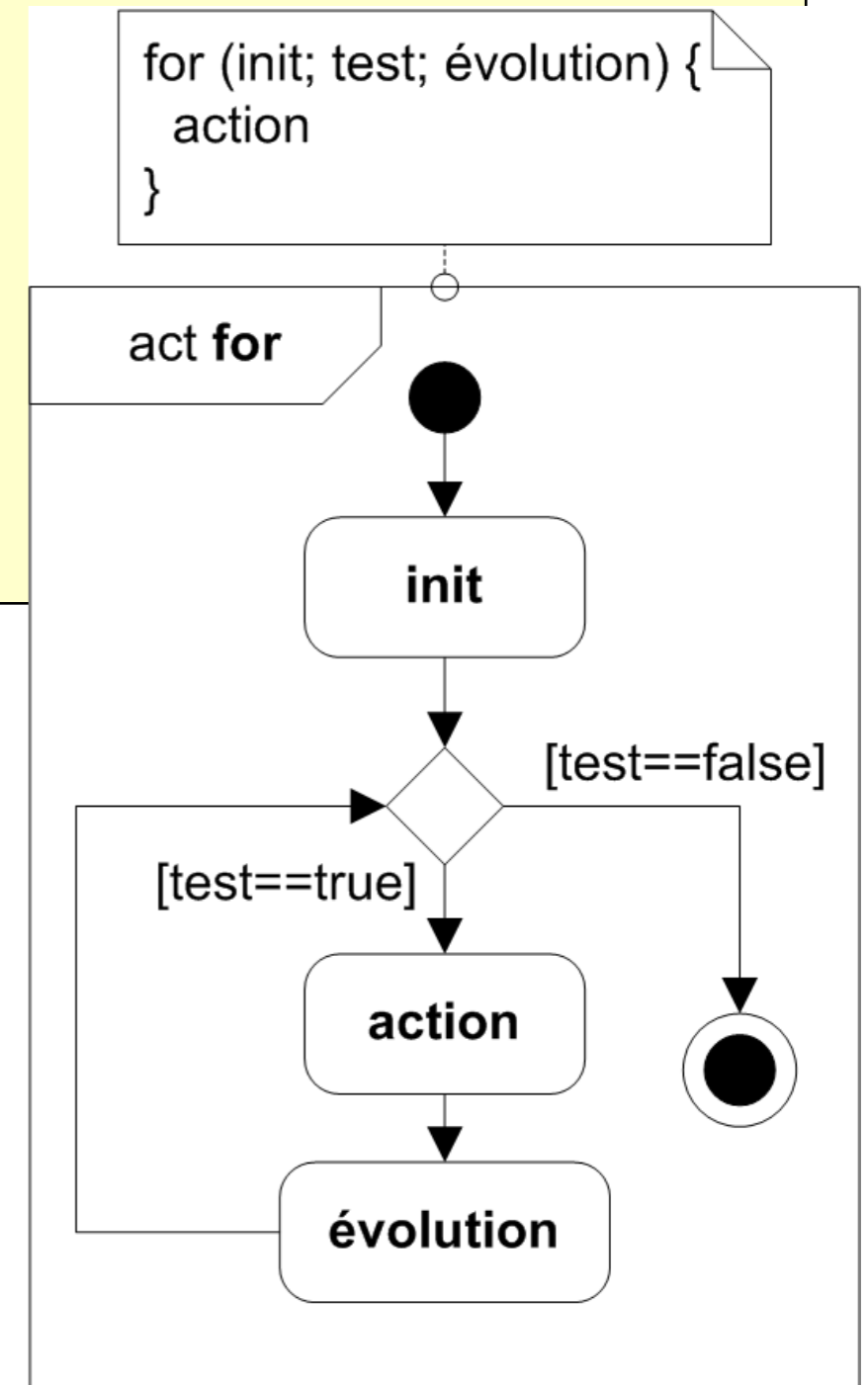
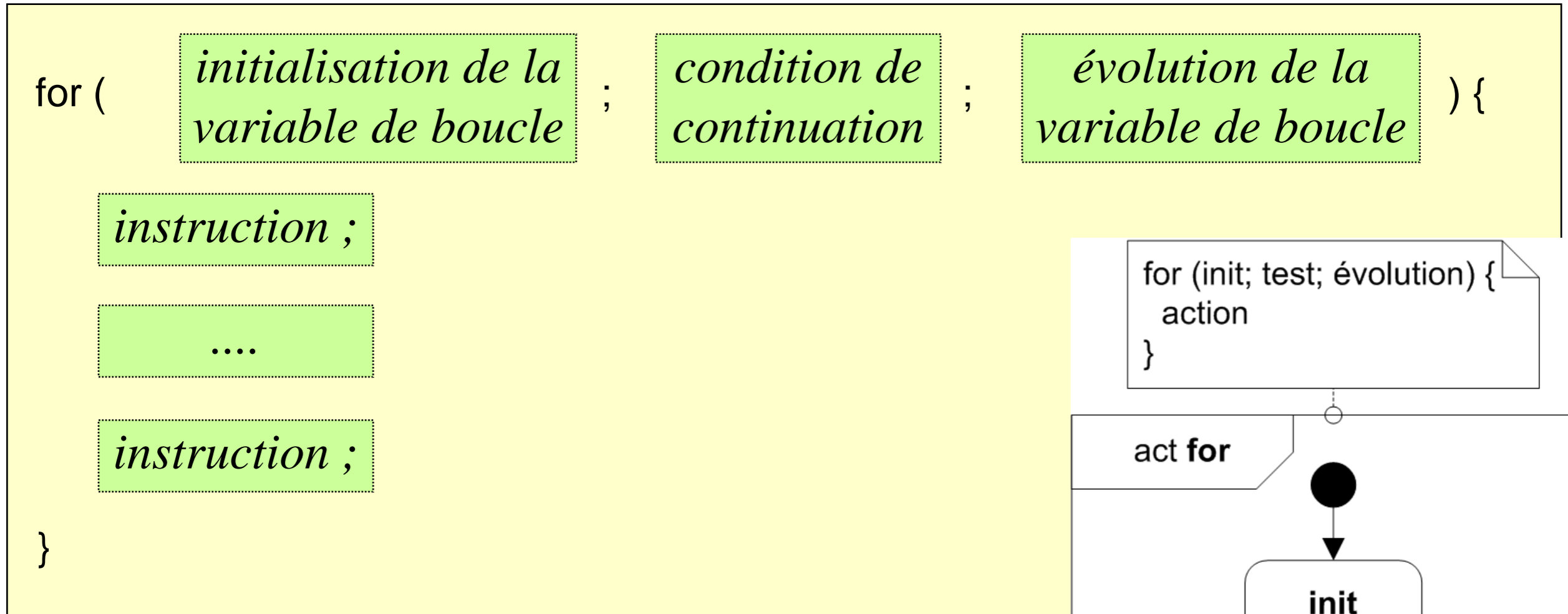
int candidat = 2;           // candidat est le candidat facteur premier courant
int exposant = 0;          // exposant représente l'exposant du facteur candidat
while (n > 1) {
    if (n % candidat == 0) { // candidat est facteur
        exposant = exposant + 1; // on augmente l'exposant
        n = n / candidat;
    } else {                // candidat n'est pas ou n'est plus facteur
        if (exposant > 0) { // Il y avait donc bien des facteurs
            print(candidat + "(" + exposant + ") "); // On les affiche !
        }
        candidat = candidat + 1; // Candidat suivant !
        exposant = 0;           // avec remise à 0 de l'exposant
    }
}
println(candidat + "(" + exposant + ") "); // Le dernier facteur en attente d'affichage !

```



# Les 3 formes de Boucles

- La **syntaxe** (forme grammaticale) de la boucle **for** est :

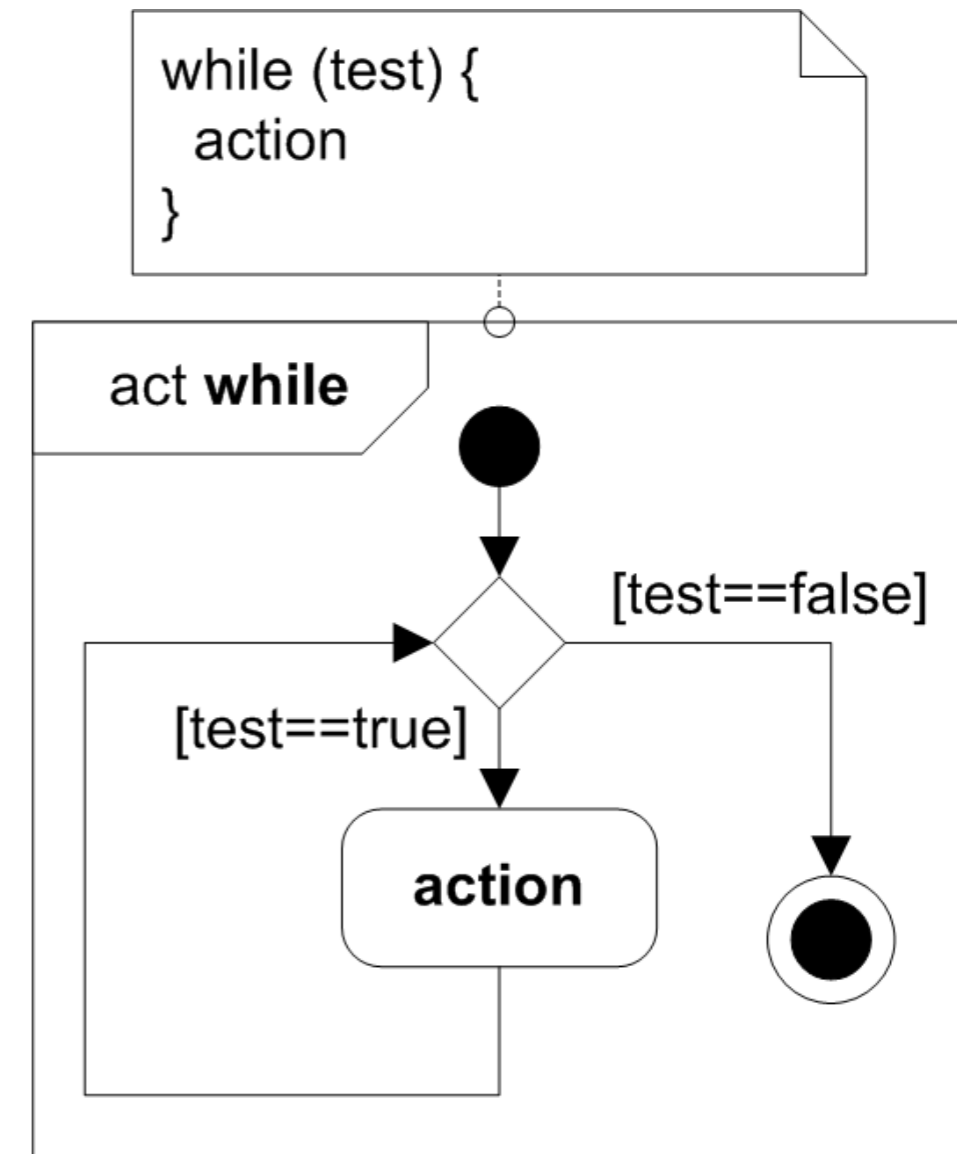
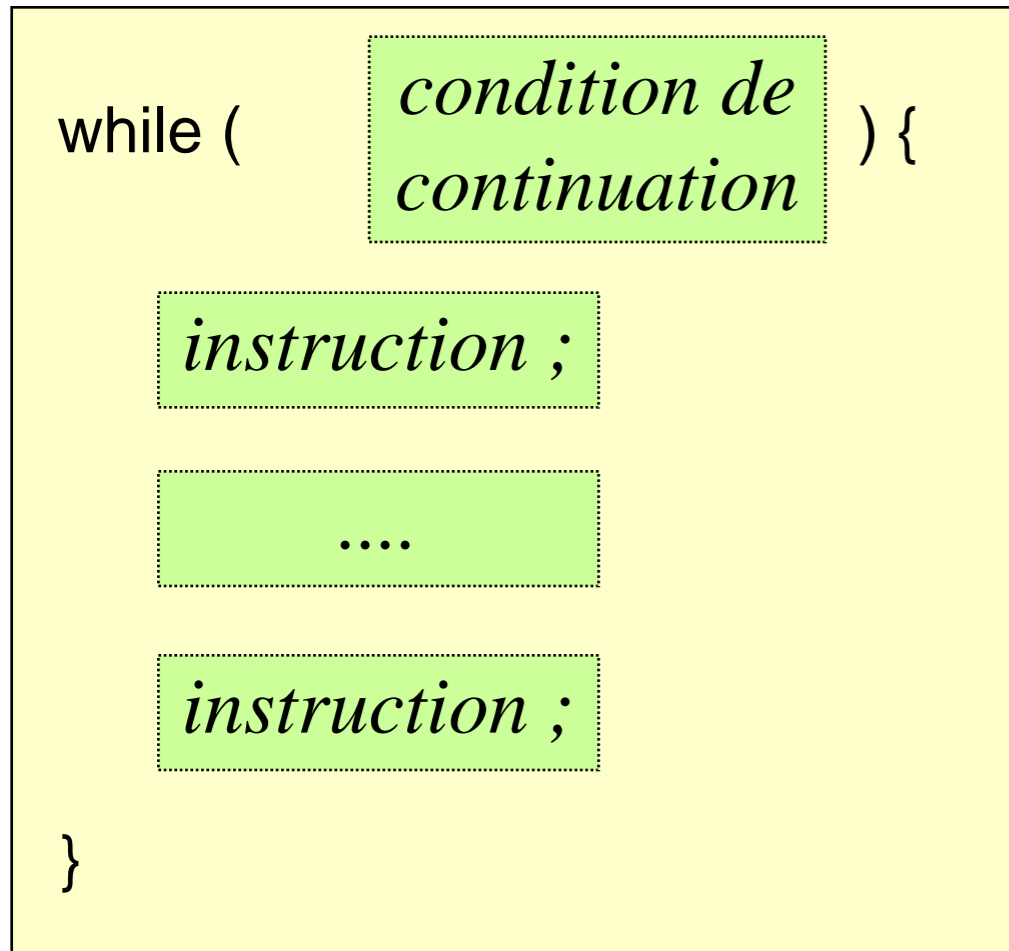


- Calcul de  $res = 1^2 + 3^2 + 5^2 + \dots + 99^2$

```

int res = 0;
for (int k = 1; k <= 99; k = k+2) {
    res = res + k * k;
}
println("1^2 + 3^2 + ... + 99^2 = " + res);
  
```

- La **syntaxe** (forme grammaticale) de la boucle **while** est :



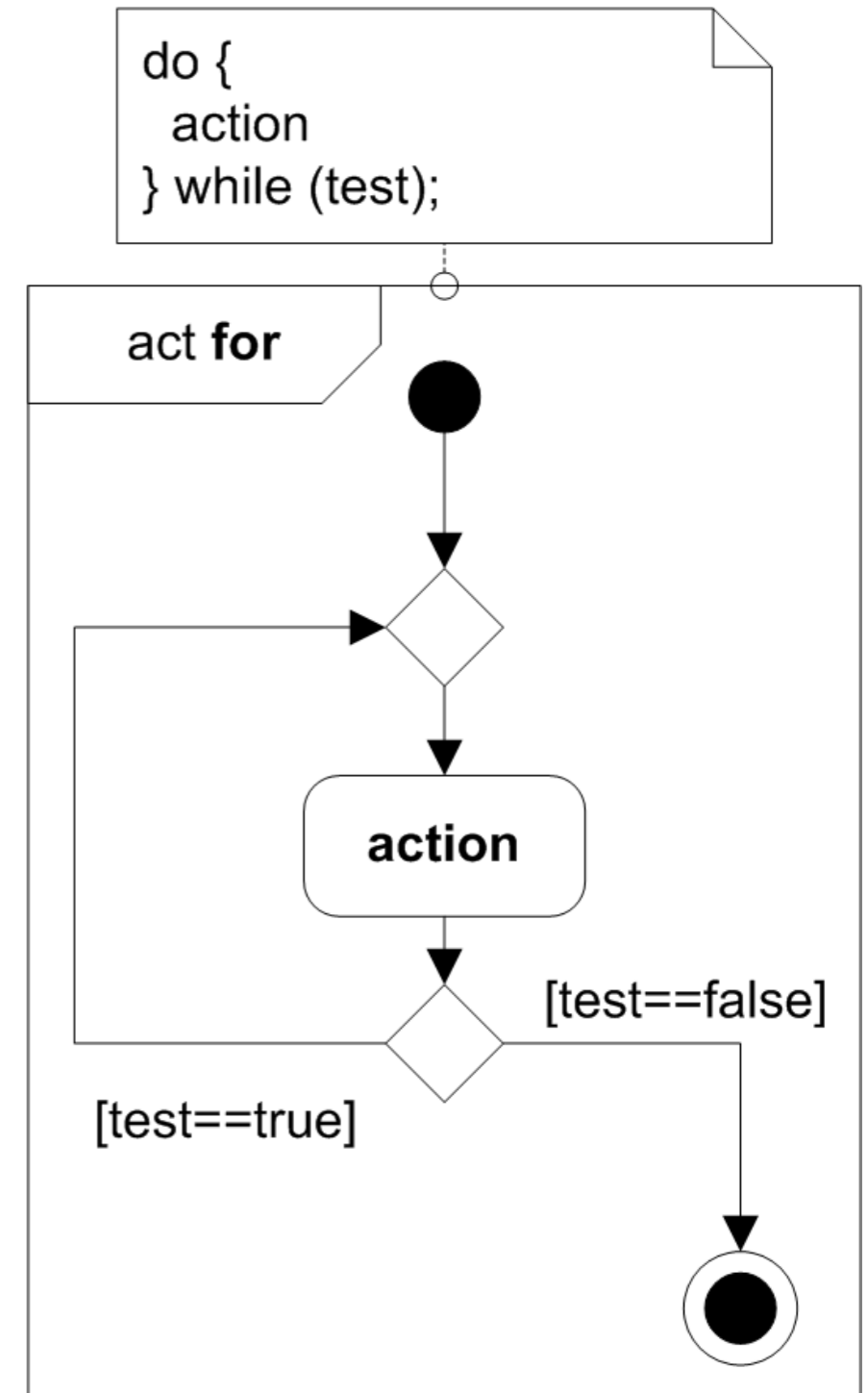
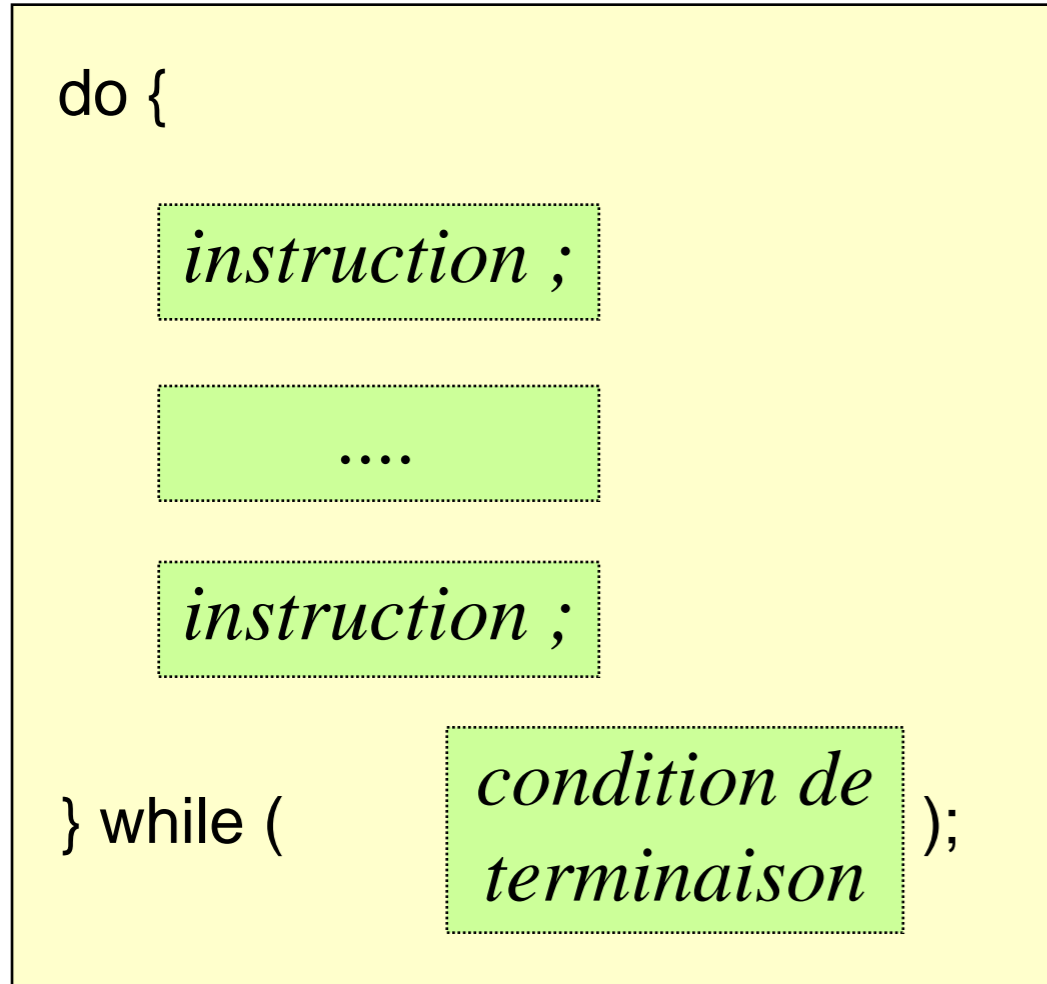
- Calcul de  $res = 1^2 + 3^2 + 5^2 + \dots + 99^2$

```

int res = 0, k = 1;
while (k <= 99) {
    res = res + k * k;
    k = k + 2;
}
println("1^2 + 3^2 + ... + 99^2 = " + res);
    
```

*Le test est effectué  
avant l'action !*

- La **syntaxe** (forme grammaticale) de la boucle **do ... while** est :



*Le test est effectué après l'action !*

- Calcul de  $res = 1^2 + 3^2 + 5^2 + \dots + 99^2$

```

int res = 0, k = 1;
do {
    res = res + k * k;
    k = k + 2;
} while (k <= 99);
println("1^2 + 3^2 + ... + 99^2 = " + res);
    
```

- **COMPLEMENT.** L'instruction `break;` permet de **casser une boucle** et d'en sortir immédiatement. L'exécution continue normalement après la boucle. Ne pas confondre avec `return;` qui permet de quitter brutalement une méthode !
- Une utilisation courante est avec une boucle for dont on s'échappe, ou bien avec une boucle infinie `while(true)`...
- Exemple : recherche du plus petit diviseur  $d \geq 2$  d'un entier  $n \geq 2$  :

```

int ppdiv(int n) {           // avec un for + break !
    int d;
    for (d = 2; d <= n; d = d + 1) {
        if (n % d == 0) {
            break;
        }
    }
    return d;
}

```

