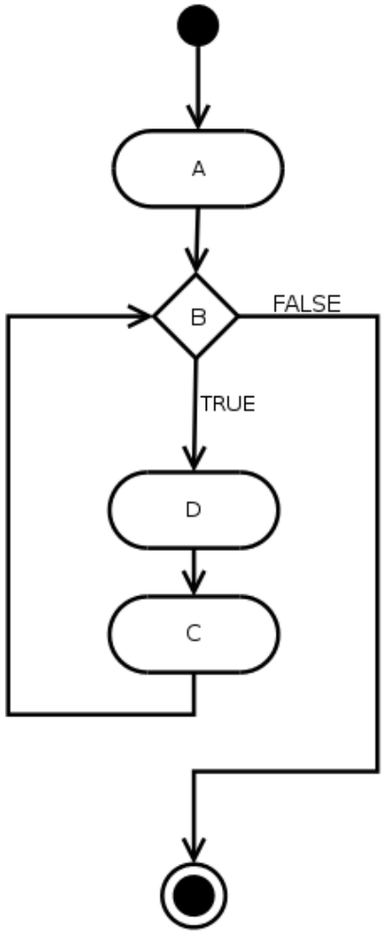
Programmation par Objets - Printemps 2010
Frédéric Mallet & Jean-Paul Roy
L1 Info&Math - Faculté des Sciences de Nice

http://deptinfo.unice.fr/~roy

Calculs répétitifs (2)

l'itération

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



La boucle **for**

- Comment calculer la factorielle n! d'un entier n ≥ 0 ?

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

$$n! = \underbrace{1 \times 2 \times \ldots \times i \times (i+1) \times \ldots \times n}$$

*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** et remplissage du résultat*

***retour** du résultat*

- Méthode **itérative** ⇔ programmée avec une boucle.

- Traduction en français et exécution du calcul de `fac(5)` :

Méthode `fac` *:* $\mathbb{N} \longrightarrow \mathbb{N}$

- *Soit* `res` *un entier initialisé à* `1`
- *Pour chaque* `i` *depuis* `2`
  tant que `i ≤ n`
  puis chaque fois `i` *est augmenté de* `1`
  multiplier `res` *par* `i`
- *Le résultat est* `res`

res	i	n
1		5
2	2	5
6	3	5
24	4	5
120	5	5
STOP	6	5

- On dit que les variables `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.

- L'une des choses qui différencient la récurrence et l'itération, c'est que dans une itération, le résultat a un nom et sa valeur à chaque ***tour de boucle*** est accessible. **Le calcul se fait pas à pas**. Dans une récurrence, les calculs en attente ne sont pas accessibles, seul le résultat final est disponible !

La boucle **while**

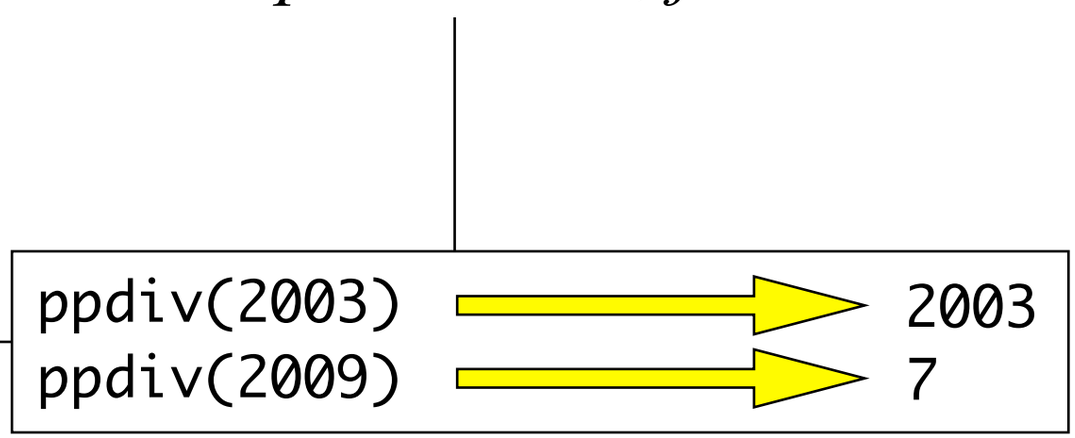
- *En principe*, dans une boucle `for`, le nombre de tours de boucle est connu à l'avance.

- 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 ≥ 2 d'un entier n ≥ 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



- 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

La variable i existe en-dehors de la boucle

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

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

⇔

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

La boucle do ... while ...

- 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 ≥ 2 d'un entier n ≥ 2.

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

```
do {
  A;
} while(B);
```

⇔

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

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 $((x_1,v_1),\ldots,(x_n,v_n))$ 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 ≥ 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 ≥ 2 que je n'ai pas encore traité.

Méthode ***afficheDiviseursPpremiers*** *:* $\mathbb{N} \longrightarrow \emptyset$

- *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 afficheDiviseursPpremiers(int n) {
  if (n < 2) return;                  // si n < 2, je ne joue pas !
  int p = 2;                          // donc n ≥ 2 et p = 2 est le premier candidat
  while (n > 1) {                     // tant qu'il reste des diviseurs non purgés
    if (n % p == 0) {                 //      si p divise n
      print(p + " ");                 //           on affiche p
      n = n / p;                      //           puis on purge p de n, et n diminue
    }
    else {                            //      sinon on avance p
      p = p + 1;
    }
  }
 println();                           // pour aller à la ligne en fin de ligne !
}
```

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

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

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

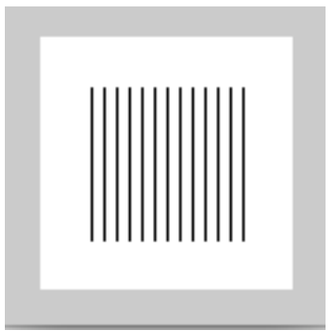
```
Décomposition de 1144660 : 2^2 5^1 11^3 43^1
```

- 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 !

```
void afficheDiviseursPremiers(int n) {
  if (n < 2) return;
  int p = 2;        // p est le candidat facteur premier courant
  int k = 0;        // k représente l'exposant du facteur p
  while (n > 1) {
    while (n % p == 0) {          // On purge complètement le facteur p
      k = k + 1;
      n = n / p;
    }
    if (k > 0) {                  // Il y avait donc bien des facteurs p
      print(p + "^" + k + " ");   // On les affiche !
    }
    p = p + 1;                    // Candidat suivant !
    k = 0;                        // avec remise à 0 de l'exposant
  }
}
```

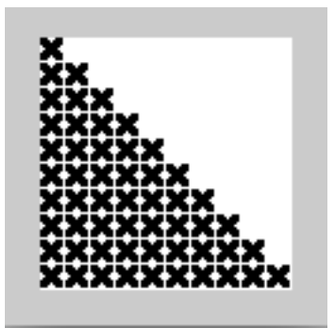
EXEMPLES GRAPHIQUES AVEC PROCESSING

```
for (int x = 20; x <= 80; x = x+5) {
  line(x,20,x,80);
}
```



```
for (float x = 80; x > 20; x = x*0.8) {
  line(20,x,80,x);
}
```

```
strokeWeight(2);
for(int y = 0; y < 100; y = y + 10) {
  for(int x = 0; x <= y; x = x + 10) {
    line(x,y,x+6,y+6);
    line(x+6,y,x,y+6);
  }
}
```



```
size(255,100);
for (int i = 0; i < 255; i = i+1) {
  stroke(255-i,0,i);
  line(i,0,i,200);
}
```

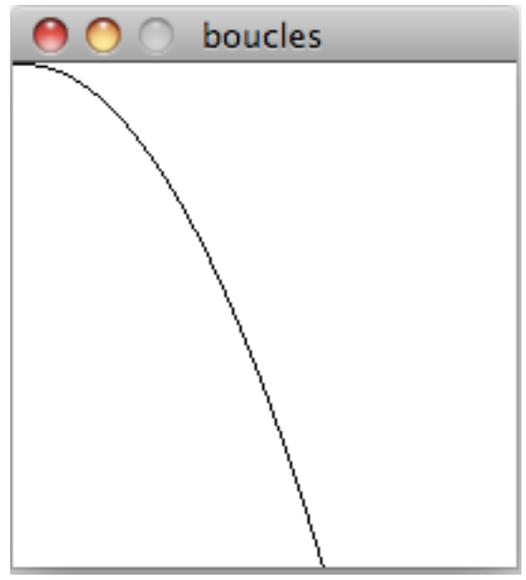
Tracé de la chute parabolique d'une bille

- Attention au repère informatique ! Regardez les axes...
- **Une bille est lancée à l'horizontale depuis le point O**, avec une vitesse initiale v_0. Dessiner sa chute gravitationnelle !
- Il s'agit de faire un dessin, pas une animation : on va *boucler* !
- La balle est attirée vers le sol par une **force verticale constante** (la pesanteur). Prenons g = 1 et masse m = 1, les unités étant arbitraires.
- L'équation fondamentale $\vec{F} = m\vec{a}$ s'écrit alors $\vec{F} = \vec{a}$ et se projette sur les deux axes en $a_x = 0$ et $a_y = 1$, d'où $v_x = Cte = v_0$ et $v_y = t$, d'où $x = v_0 t$ et $y = 0.5\ t^2$. Il suffit donc de faire varier t.

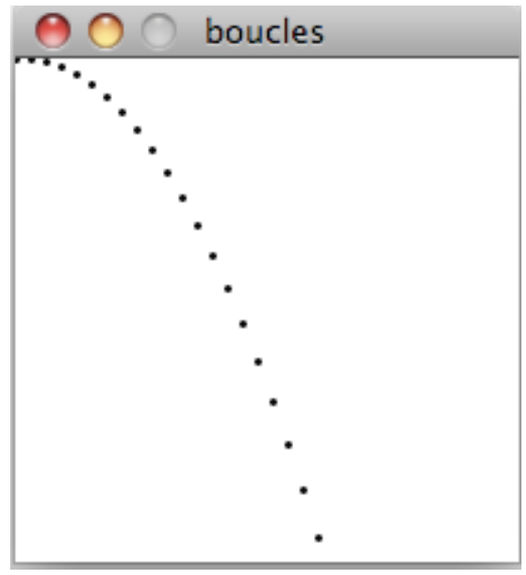
$$\begin{pmatrix} x = v_0 t \\ y = 0.5t^2 \end{pmatrix} \text{ et } \begin{pmatrix} dx = v_0 \\ dy = t \end{pmatrix} \qquad dt = 1$$

- La courbe résultante de la trajectoire n'est autre qu'une suite de petits segments joignant les points (x(t),y(t)) et (x(t+1),y(t+1)).

```
void chuteParabolique1() {      // on relie les points de la courbe par des segments
  size(200,200);
  background(255,255,255);
  strokeWeight(3);
  smooth();
  float v0 = 6, x = 0, y = 0;
  for (int t = 0; x < 200; t = t + 1) {
    float xSuiv = x + v0;
    float ySuiv = y + t;
    line(x,y,xSuiv,ySuiv);
    x = xSuiv;
    y = ySuiv;
  }
}
```

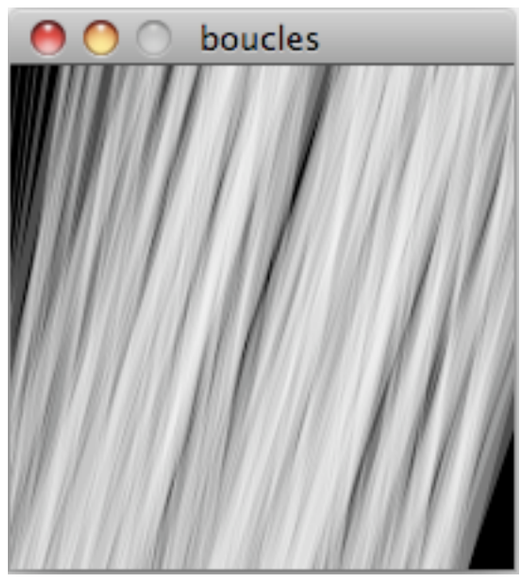


```
void chuteParabolique2() {      // uniquement les points de la courbe
  size(200,200);
  background(255,255,255);
  strokeWeight(3);
  float v0 = 6, x = 0, y = 0;
  for (int t = 0; x < 200; t = t + 1) {
    point(x,y);
    x = x + v0;
    y =  y + t;
  }
}
```

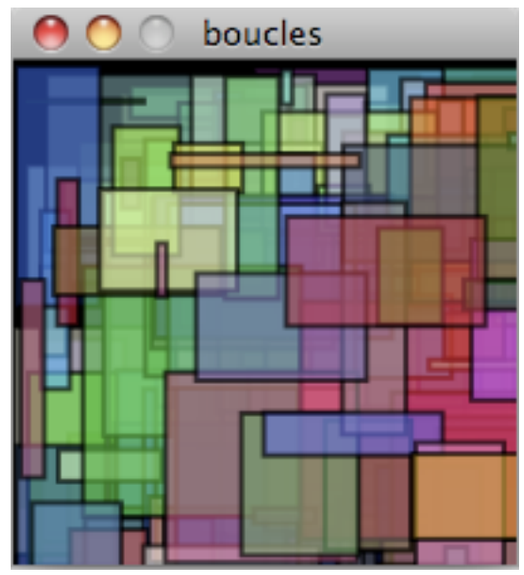


Dessiner avec l'aléatoire

```
void aleatoire1() {
  size(200,200);
  background(0,0,0);
  stroke(255,255,255,60);     // 60 = transparence
  for (int i = 0; i < 200; i = i+1) {
    float r = random(10);
    strokeWeight(r);
    float ecart = 5 * r;
    line(i - 20, 200, i + ecart, 0);
  }
}
```



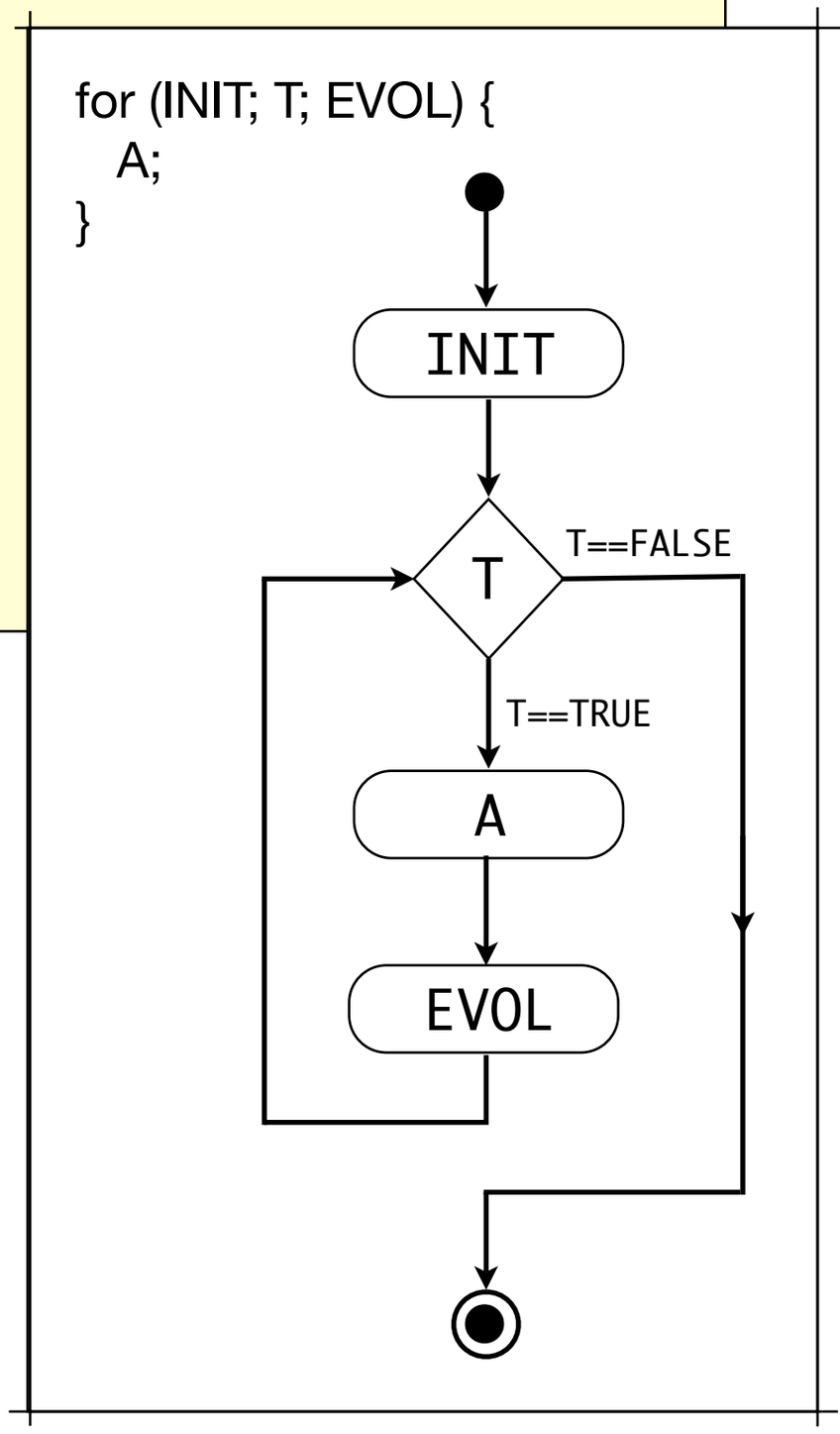

```
void aleatoire2() {
  size(200,200);
  background(0,0,0);
  stroke(0,0,0,180);
  strokeWeight(2);
  for (int k = 0; k < 200; k = k+1) {
    fill(random(255),random(255),random(255),150);
    rect(random(200),random(200),random(100),random(100));
  }
}
```



Les 3 formes de Boucles

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

```
for ( initialisation de la variable de boucle ; condition de continuation ; évolution de la variable de boucle ) {
    instruction ;
    ....
    instruction ;
}
```

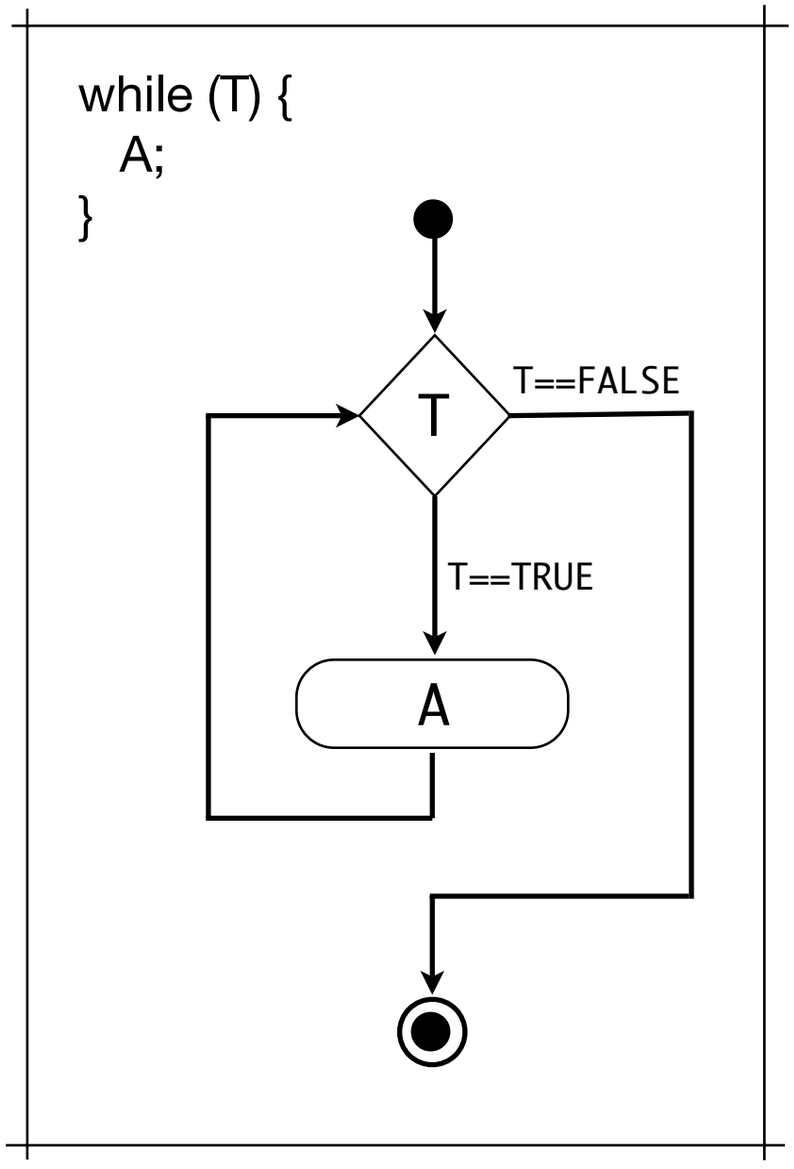


- Calcul de res = $1^2 + 3^2 + 5^2 + \ldots + 99^2$

```
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 :

```
while ( condition de continuation ) {
    instruction ;
    ....
    instruction ;
}
```



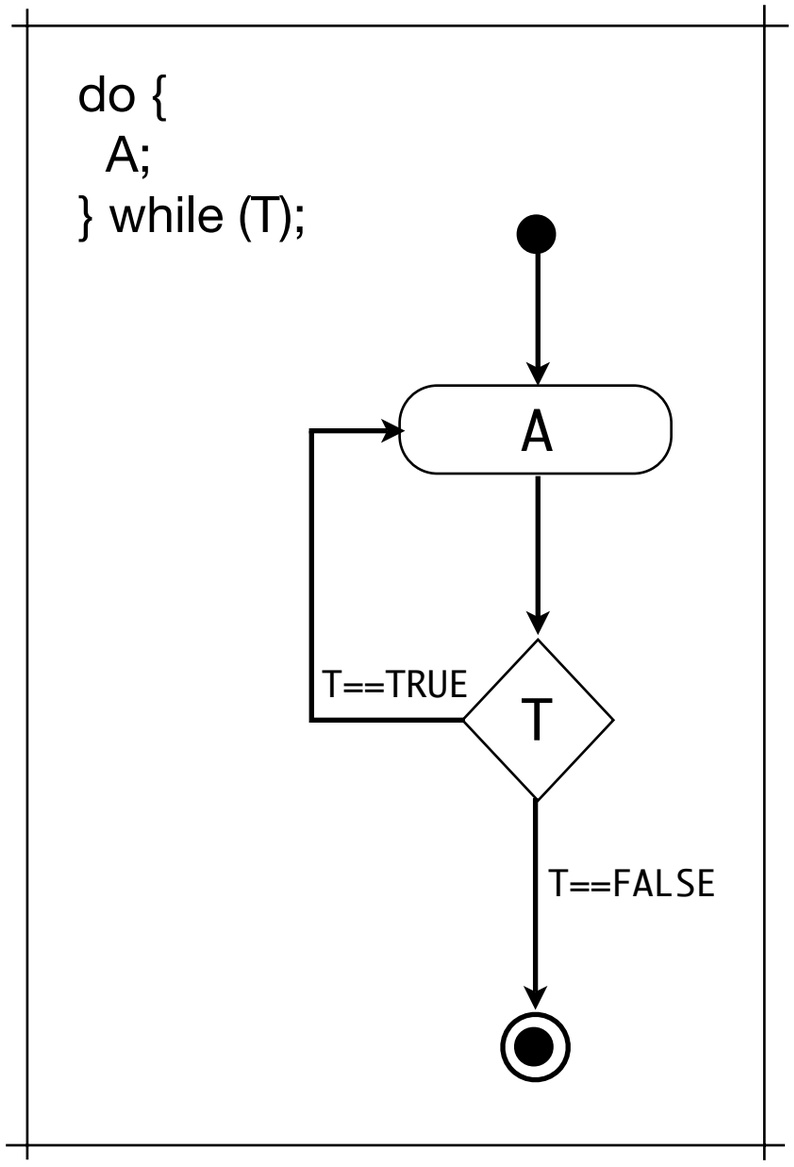
- Calcul de `res` = $1^2 + 3^2 + 5^2 + \ldots + 99^2$

```
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 :

```
do {
    instruction ;
    ....
    instruction ;
} while ( condition de terminaison );
```



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

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

Le test est effectué ***après*** *l'action !*

• ***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 ≥ 2 d'un entier n ≥ 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;
}
```