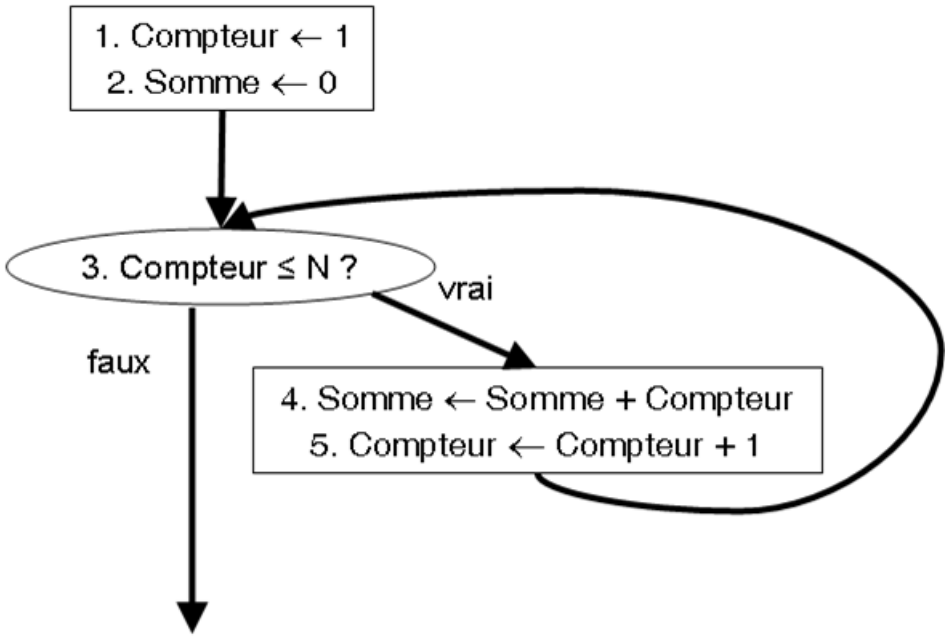
ITI1520 - Section 6 Solutionnaire aux exercices

Mémoire de programme **Exercice 6-1 - Somme de 1 à N** **Mémoire de travail**

DONNÉES: N (un entier positif)
INTERMÉDIAIRES: Compteur (compteur de 1 à N)
RÉSULTATS: Somme (somme des entiers de 1 à N)
EN-TÊTE: Somme ← Somme1àN(N)
MODULE:



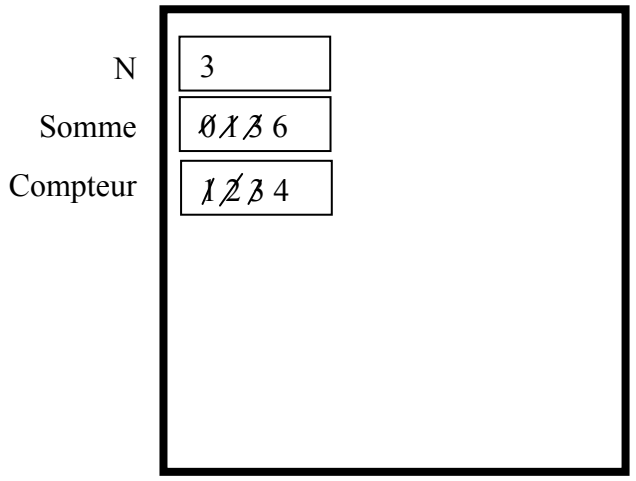
N
Somme
Compteur

UCT

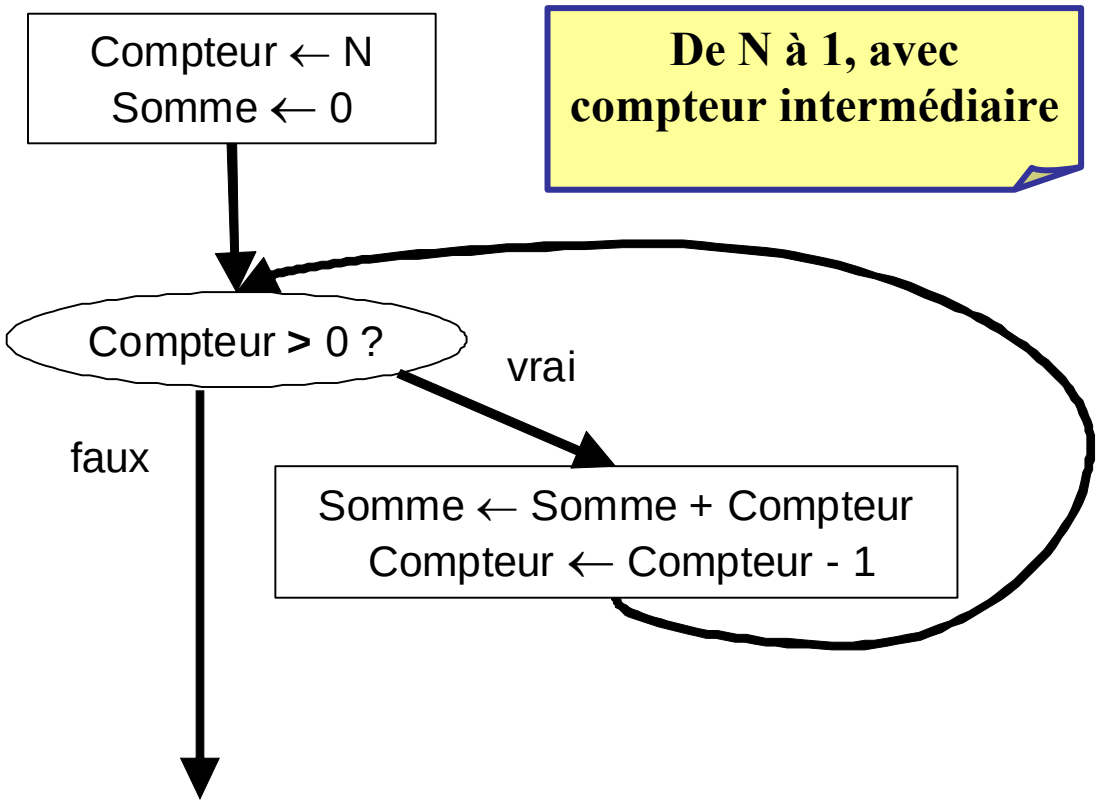
Exercice 6-1: Trace de Somme1àN(3)

	N	Compteur	Somme
Init.	**3**	**?**	**?**
1.		**1**	
2.			**0**
3. VRAI			
4.			**1**
5.		**2**	
3. VRAI			
4.			**3**
5.		**3**	
3. VRAI			
4.			**6**
5.		**4**	
3. FAUX			

Mémoire de travail

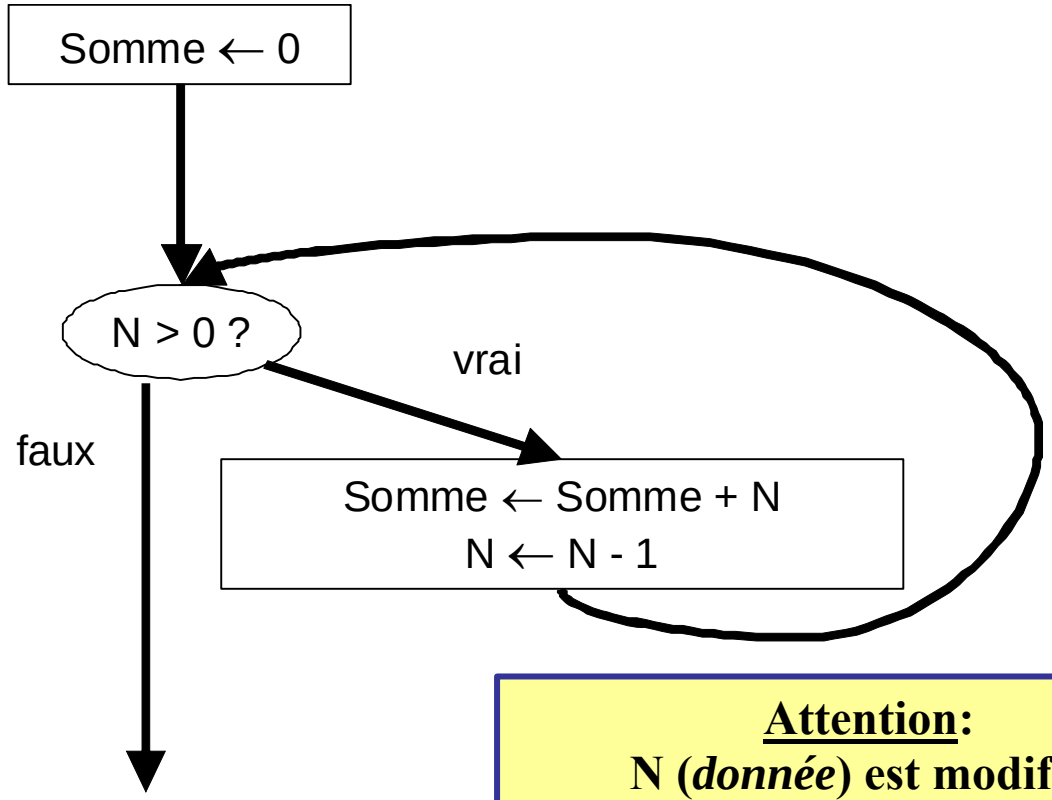


DONNÉES: N (un entier positif)
INTERMÉDIAIRES: Compteur (compteur de N à 1)
RÉSULTATS: Somme (somme des entiers de 1 à N)
EN-TÊTE: Somme ← Somme1àN(N)
MODULE:



UCT

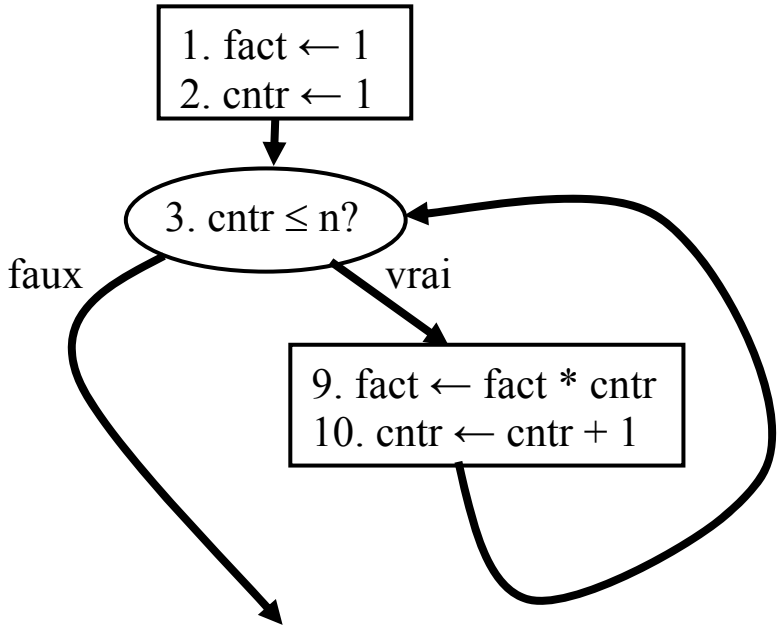
DONNÉES: N (un entier positif)
INTERMÉDIAIRES: (aucune)
RÉSULTATS: Somme (somme des entiers de 1 à N)
EN-TÊTE: Somme ← Somme1àN(N)
MODULE:



<u>Attention</u>:
N (*donnée*) est modifiée *localement* seulement. L'approche fonctionne, mais peut porter à confusion.

UCT

DONNÉES: n (nombre pour calculer factoriel)
RÉSULTATS: fact (n factoriel)
INTERMÉDIAIRES: cntr (pour compter de 1 à N)
HYPOTHÈSES: n devrait être positif
EN-TÊTE: fact ← factoriel (n)
MODULE:



UCT

DONNÉES:

unNombre (nombre)

diviseur (diviseur – sert de base du log)

RÉSULTATS:

partEnt (partie entière du logarithme, nombre de fois que unNombre est divisible par diviseur)

INTERMÉDIAIRES:

quotient (quotient de la division)

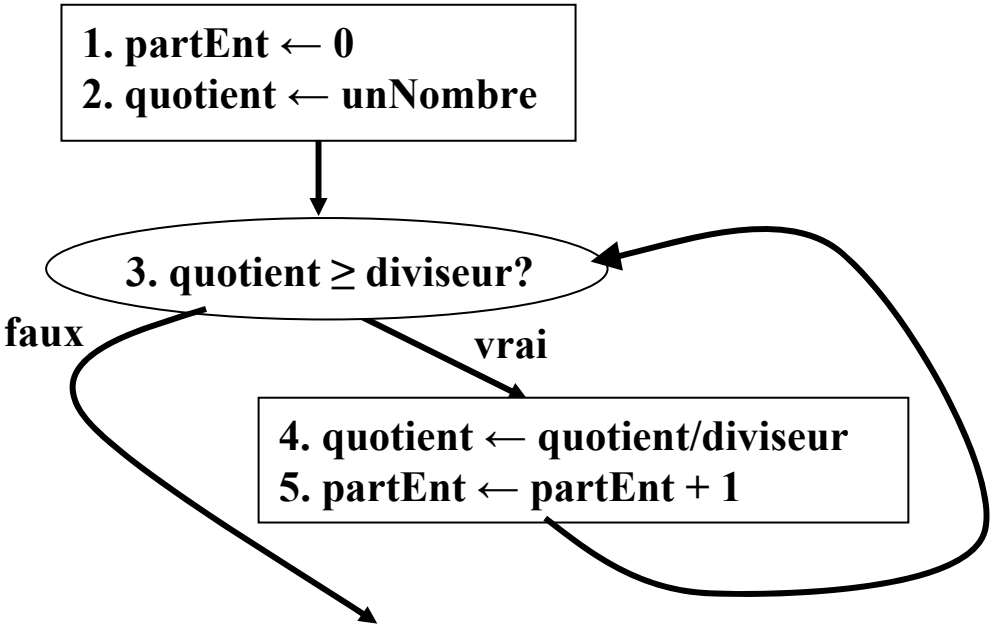
HYPOTHÈSES:

(unNombre devrait être positif)

EN-TÊTE:

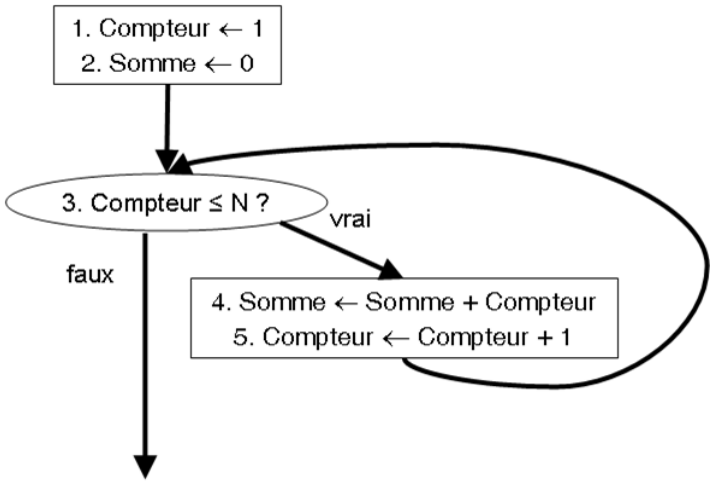
partEnt ← logInt (unNombre, diviseur)

MODULE:



UCT

DONNÉES: N (un entier positif)
INTERMÉDIAIRES: Compteur (compteur de 1 à N)
RÉSULTATS: Somme (somme des entiers de 1 à N)
EN-TÊTE: Somme ← Somme1àN(N)
MODULE:

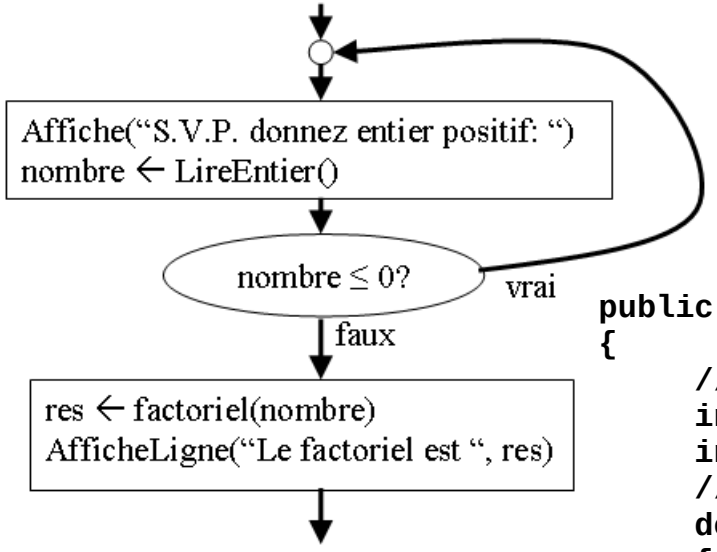



```
public static int somme1àN (int n)
{
      int compteur;
      int somme;

      // MODULE DE L'ALGORITHME
      compteur = 1;
      somme = 0;
      // Boucle
      while (compteur <= n)
      {
         somme = somme + compteur ;
         compteur = compteur + 1 ;
      }
      // Retour des résultats
      return(n);
}
```

UCT

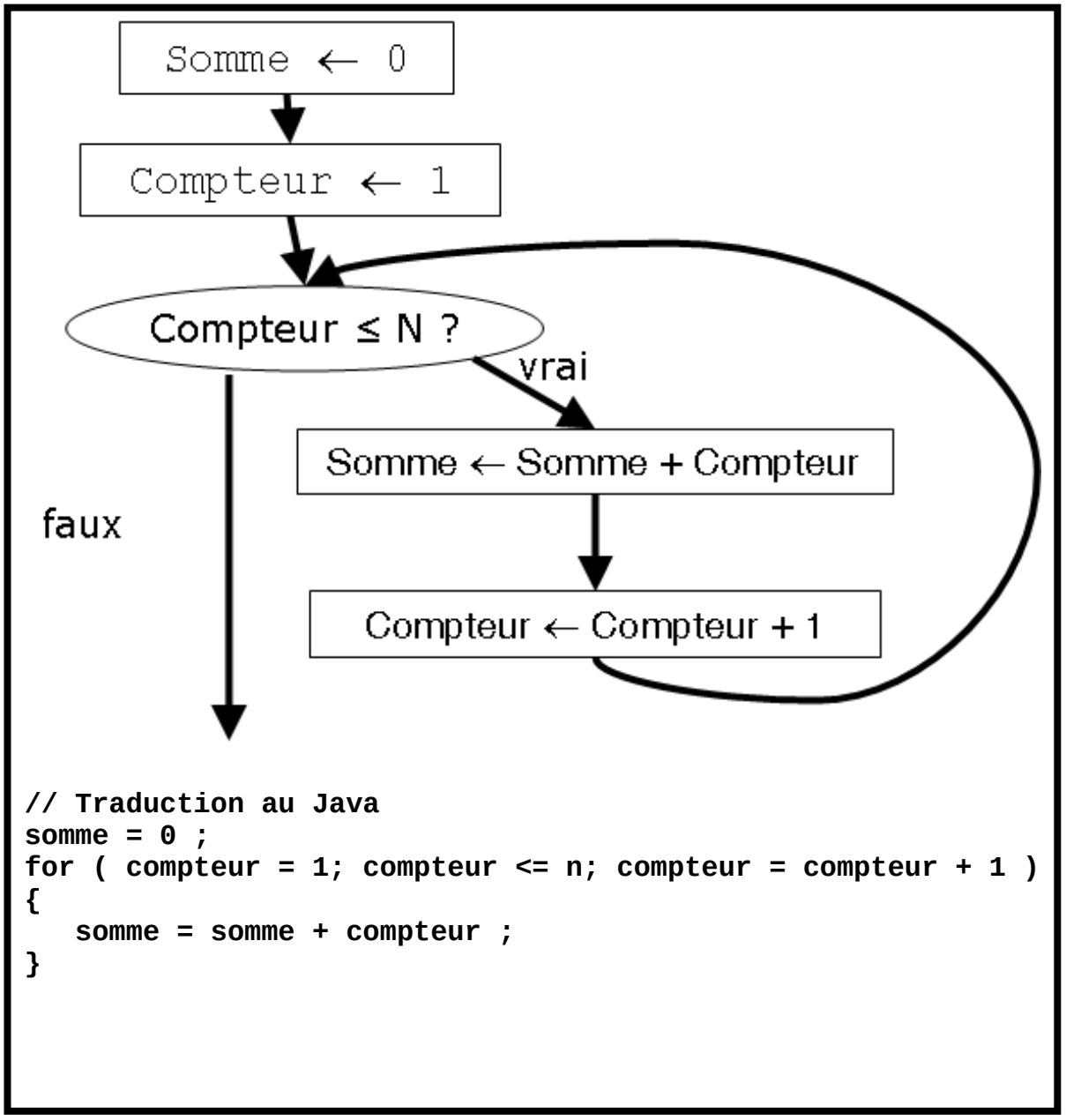
DONNÉES: (aucune)
RÉSULTATS: (aucune)
INTERMÉDIAIRES: nombre (nombre pour calculer factoriel)
res (le factoriel, i.e. nombre!)
CONSTRAINTS: (nombre devrait être positif)
EN-TÊTE: Principal()
MODULE:




```
public static void main(String args)
{
     // Variables
     int nombre; // nombre donné par utilisateur
     int res; // factoriel, i.e. nombre!
     // Module
     do // Boucle post-test
     {
          System.out.print(
               "S.V.P  donnez un entier positif: ");
          nombre = ITI1520.ReadInt();
     } while(nombre <= 0);
     res = factoriel(nombre);
     System.out.println("Le factoriel est "
                        + nombre);
}
```

UCT

Mémoire de programme **Exercice 6-6 – Somme1àN avec boucle FOR** **Mémoire de travail**




```
// Traduction au Java
somme = 0 ;
for ( compteur = 1; compteur <= n; compteur = compteur + 1 )
{
   somme = somme + compteur ;
}
```

UCT

Exercice 6-7 – Utilisation des indices

- L'indice d'un tableau T de taille L peut être n'importe quelle expression mathématique retournant un entier entre 0 et L-1.
  - Soit K =2, et T réfère à

2	-1	5	7

(de longueur 4)

T[2] = **5**
T[K] = **5**
T[2*K-1] = **7**
T[T[0]+1] = **7**

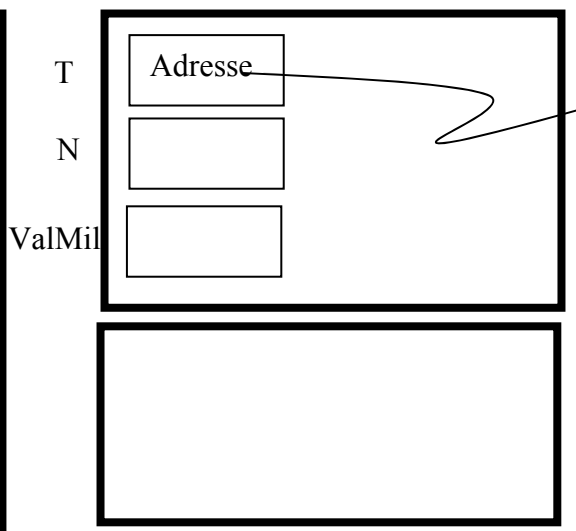
- T[expression] peut être utilisé comme toute autre variable dans des expressions plus complexes.
- **Rappelez-vous T est une variable de référence**

Mémoire de programme

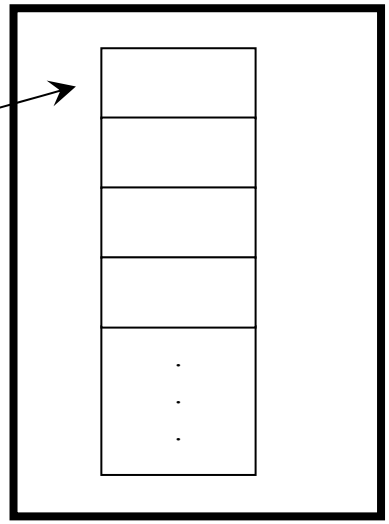
Exercice 6-8 – Valeur au milieu d'un tableau

DONNÉES: T (réfère à un tableau de nombres réels)
N (un entier impair, la longueur du tableau)
RÉSULTATS: ValMil (nombre au milieu du tableau
INTERMÉDIAIRES: **(aucune)**
EN-TÊTE: **ValMil← milieu (T, N)**
MODULE
ValMil← T[(N – 1) / 2]

Mémoire de travail



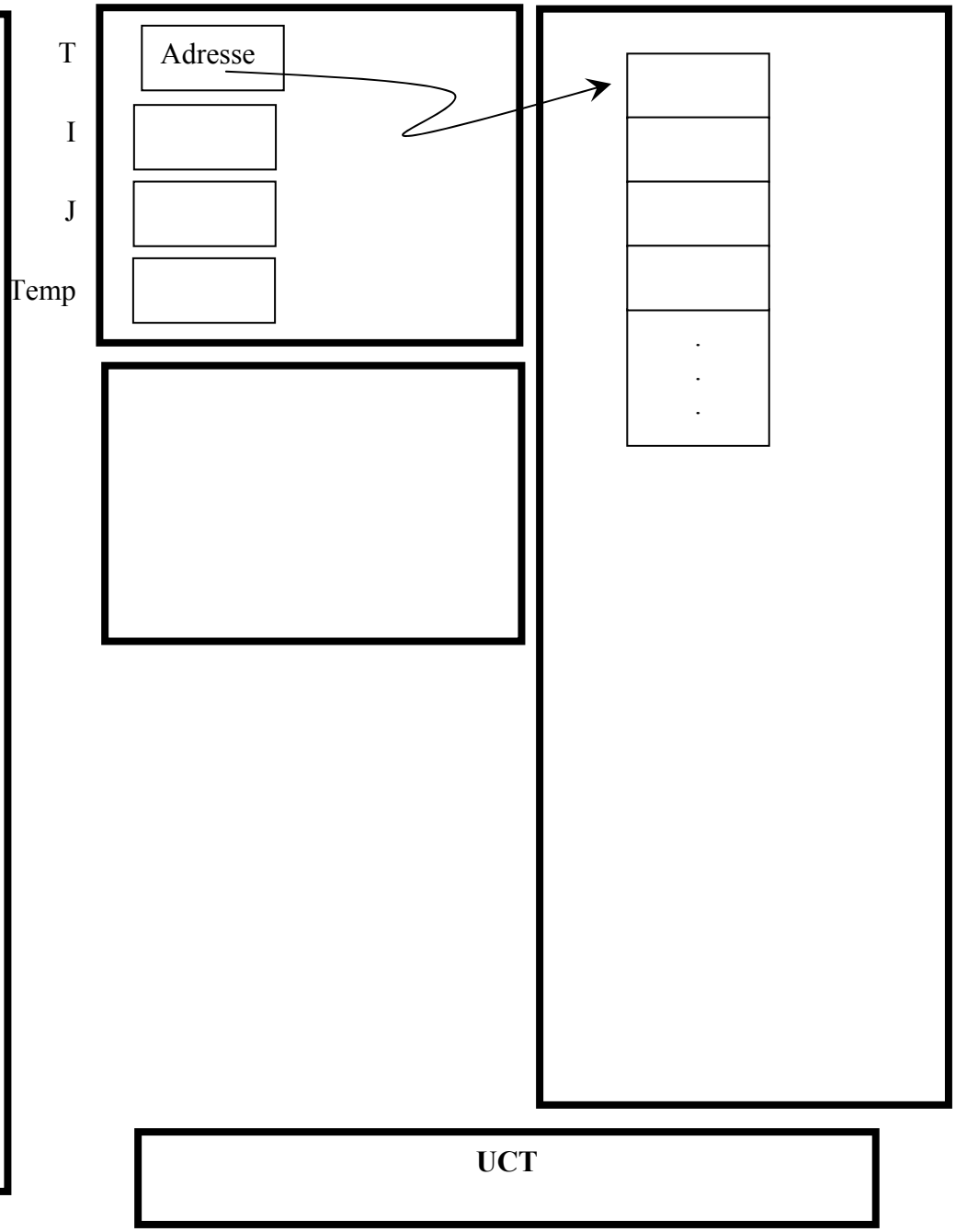
Mémoire globale



UCT

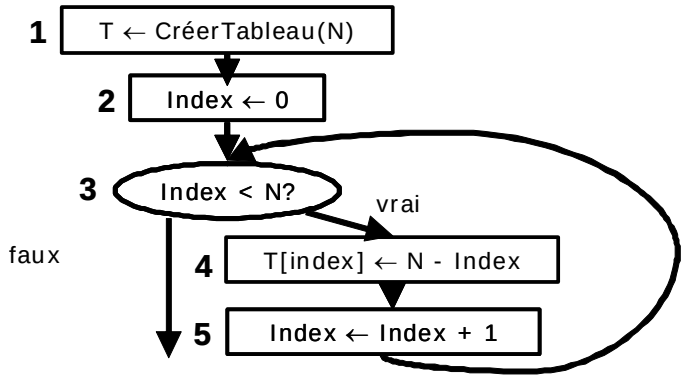
Mémoire de programme **Exercice 6-9 – Échanger les valeurs d'un tableau** **Mémoire de travail** **Mémoire globale**

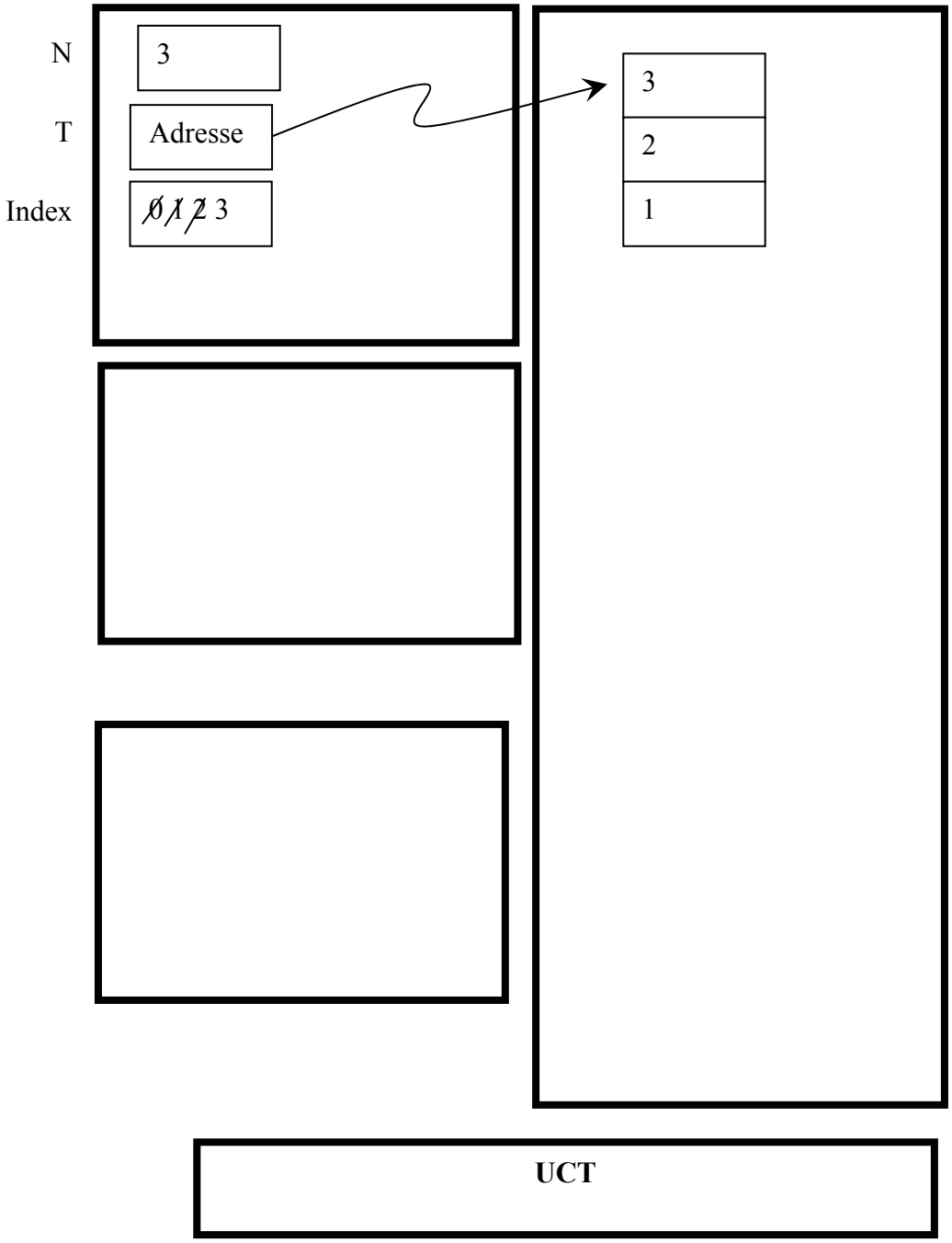
DONNÉES: T (réfère à un tableau d'entiers)
I, J (deux indices)
MODIFIÉS: **T (avec T[I] et T[J] échangés)**
INTERMÉDIAIRES:
Temp (conserve T[I] temporairement)
RÉSULTATS: **(aucun)**
EN-TÊTE: Échange(T, I, J)
MODULE
Temp ← T[I]
T[I] ← T[J]
T[J] ← Temp



Mémoire de programme **Exercice 6-10 – Création d'un nouveau tableau** **Mémoire de travail** **Mémoire globale**

DONNÉES: N (un entier positif)
RÉSULTATS: T (une référence à un tableau contenant N…1)
INTERMÉDIAIRES: **Index (un index pour T)**
EN-TÊTE: **T ← CréeNà1(N)**
MODULE:

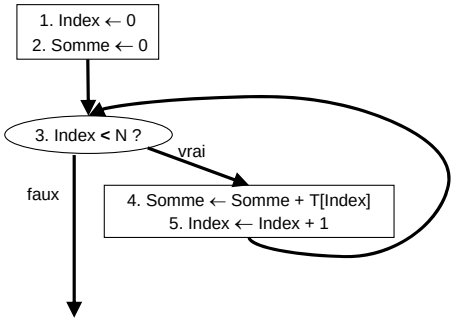




Trace pour N=3

Instructions	N	Index	T
Valeurs initiales	3	?	?
1. T ← CréerTableau(3)			{?, ?, ?}
2. Index ← 0		0	
3. Index < N? vrai			
4. T[Index] ← N – Index			{3,?,?}
5. Index ← Index + 1		1	
3. Index < N? vrai			
4. T[Index] ← N – Index			{3,2,?}
5. Index ← Index + 1		2	
3. Index < N? vrai			
4. T[Index] ← N – Index			{3,2,1}
5. Index ← Index + 1		3	
3. Index < N? faux			

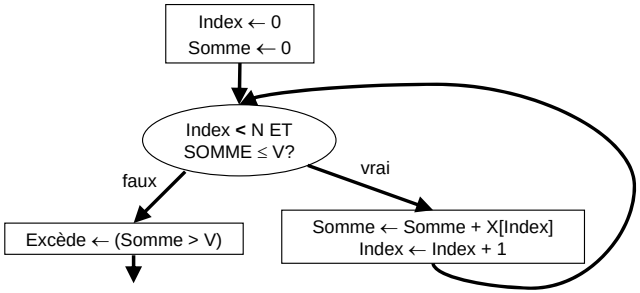
Modèle algorithmique	Java
Exercice 6-11 - Somme des valeurs d'un tableau	

DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
INTERMÉDIAIRES: Index (Index de tableau allant de 0 to N-1)
RÉSULTATS: Somme (Somme du contenu du tableau)
EN-TÊTE: Somme ← SommeTableau(T,N)
MODULE:
 
 	
	`public static int[] sommeTableau(int[] t)`
	`{`
	`  // notez, donnée n est t.length`
	`  int n=t.length;`
	`  // Résultats`
	`  int somme;`
	`  // Intermédiaires`
	`  int index;`
	`  //`
	`  somme = 0;  // initialise somme`
	`  // boucle for`
	`  for(index = 0; index < n; index++)`
	`  {`
	`  somme = somme + t[index];`
	`  }`
	`  // retourner les résultats`
	`  return(somme);`
	`}`

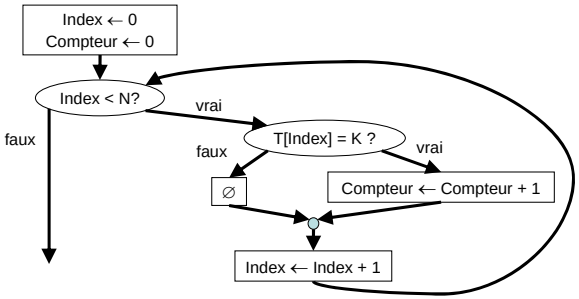
Exercice 6-12 (a) - Soit une valeur V et un tableau X de N valeurs, vérifiez si la somme des valeurs de X excède V.	

DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
V (Une valeur limite)
INTERMÉDIAIRES: Somme (La somme des valeurs, de l'exemple 2)
RÉSULTATS: Excède (Booléen: Vrai si Somme > V sinon faux)
EN-TÊTE: Excède ← SommeExcèdeT(T,N,V)
MODULE:
Somme ← SommeTableau(T,N)
Excède ← Somme > V
	`public static boolean sommeExcède(int[] t, int v)`
	`{`
	`  // notez, donné n est t.length`
	`  // Résultats`
	`  int excède;`
	`  // Intermédiaires`
	`  int somme;`
	`  //`
	`  somme = sommeTableau(t);  // obtenir somme`
	`  excède = somme > t;`
	`  // retourner les résultats`
	`  return(excède);`
	`}`

Modèle algorithmique	Java
Exercice 6-12 (b) - Soit une valeur V et un tableau X de N valeurs, vérifiez si la somme des valeurs de X excède V.	

DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
V (Une valeur limite)
INTERMÉDIAIRES: Index (index du tableau X allant de 0 à N-1)
Somme (La somme des valeurs, de l'exemple 2)
RÉSULTATS: Excède (Booléen: Vrai si Somme > V sinon faux)
EN-TÊTE: Excède ← SommeExcèdeT(T,N,V)
MODULE:
 
 	
	`public static boolean sommeExcèdeT (int[] t, int v)`
	`{`
	`  // notez, donnée n est t.length`
	`  int n = t.length;`
	`  // Résultats`
	`  int excède;`
	`  // intermédiaire`
	`  int index`
	`  int somme;`
	`  // boucle avec while`
	`  somme = 0;`
	`  index = 0;`
	`  while(index < n && somme <= t)`
	`  {`
	`  somme = somme+t[index];`
	`  index = index+1;`
	`  }`
	`  excède = somme > t;`
	`  // retourner les résultats`
	`  return(excède);`
	`}`

Exercice 6-13 - Comptez combien de fois K apparaît dans un tableau de taille N.	

DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
K (valeur dont on va compter les instances)
INTERMÉDIAIRES:
Index (Index de tableau de 0 à N-1)
RÉSULTATS: Compteur (Nombres d'intances de K dans T
EN-TÊTE: Compteur ← CompteK(T,N,K)
MODULE:
 
 	
	`public static int compteK(int[] t, int k)`
	`{`
	`  // Résultats`
	`  int compteur;`
	`  // Intermédiaires`
	`  int index;`
	`  // initialise compteur`
	`  compteur = 0;`
	`  for(index = 0; index < t.length; index=index+1) // boucle`
	`  {`
	`  if(t[index] == k)`
	`  {`
	`  compteur = compteur + 1;`
	`  }`
	`  else { /* rien faire */ }`
	`  }`
	`  return(compteur); // retourner les résultats`
	`}`

Exercice 6-14 (a) - Soit un tableau T de N valeurs et un nombre K, vérifiez si K apparaît dans T ou non.

Modèle algorithmique

DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
K (Valeur recherchée dans tableau)
INTERMÉDIAIRES:
Compteur (Nombres d'instances de K dans tableau, de l'algorithme 4)
RÉSULTATS: Trouvé (Booléen: vrai si K est dans tableau, sinon faux
EN-TÊTE: Trouvé ← TrouveK(T,N,K)
MODULE:
Compteur ← CompteK(T,N,K)
Trouvé ← Compteur > 0

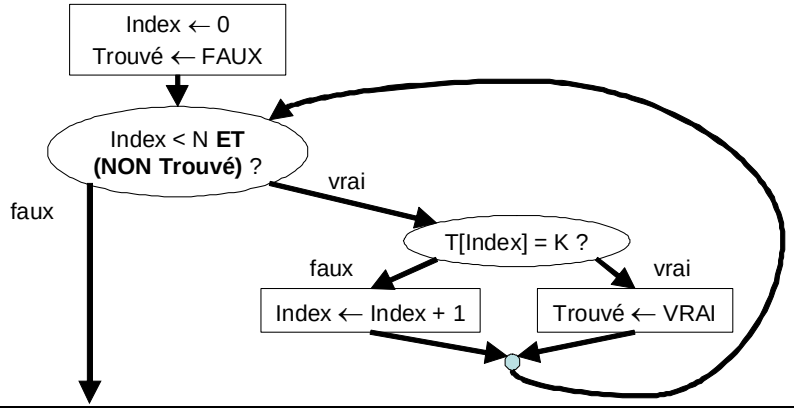
Java

```
public static boolean trouveK(int[] t, int k)
{
  // Résultats
  boolean trouvé;
  // Intermédiaires
  int compteur;
  // Module
  compteur = compteK(t, k);
  trouvé = compteur > 0;
  // Retourner résultat
  return(trouvé);
}
```

Exercice 6-14 (b) - Soit un tableau T de N valeurs et un nombre K, vérifiez si K apparaît dans T ou non.

Modèle algorithmique

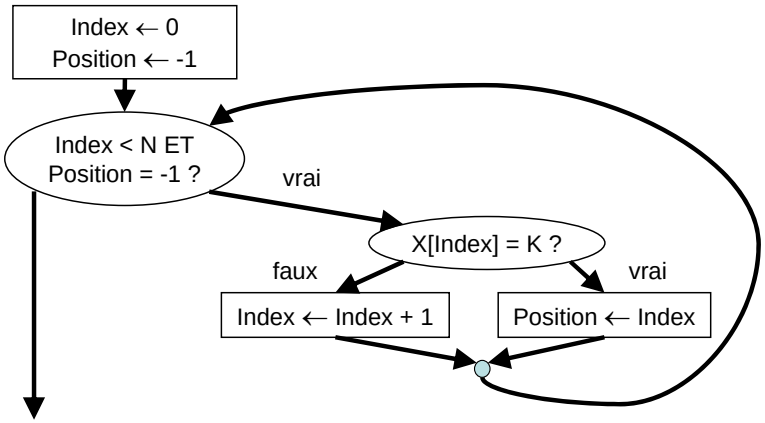
DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
K (Valeur recherchée dans tableau)
INTERMÉDIAIRES:
Index (index de T allant de 0 à N-1)
RÉSULTATS: Trouvé (Booléen: vrai si K est dans tableau, sinon faux)
EN-TÊTE: Trouvé ← TrouveK(T,N,K)
MODULE:



Java

```
public static boolean trouveK(int[] t, int k)
{
   boolean trouvé; // résultats
   int index; // Intermédiaires
   // initialise trouvé and index
   trouvé = false;
   index = 0;
   while(index < t.length && !trouvé) // boucle
   {
      if(t[index] == k)
      {
        trouvé = true;
      }
      else { /* rien faire */ }
      index=index+1;
   }
   return(trouvé); // retourner les résultats
}
```

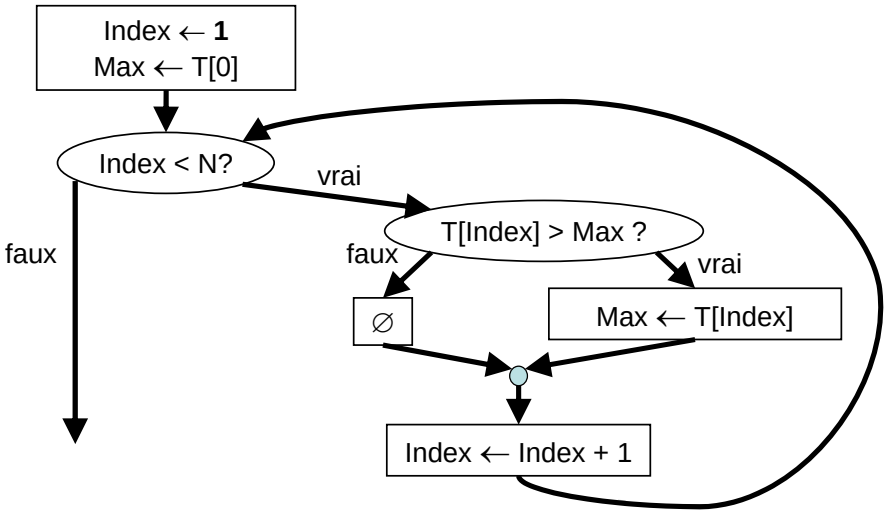
Modèle algorithmique	Java
Exercice 6-15 - Soit un tableau X de N valeurs et un nombre K, trouvez la position de la première occurrence de K. (Si K n'est pas dans X, retournez –1 comme position.)	

DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
K (Valeur recherchée dans tableau)
INTERMÉDIAIRES:
Index (index de T allant de 0 à N-1)
RÉSULTATS: Position (Position de K dans tableau, ou -1 si K n'est pas dans le tableau)
EN-TÊTE: Position ← OùEstK(T,N,K)
MODULE:
 
 	
	`public static int oùEstK(int t[], int n, int k)`
	`{`
	`int position; // Résultat`
	`int index; //Intermédiaire`
	`// Module`
	`index = 0;`
	`position = -1;`
	`while( (index < n) && (position == -1) )`
	`{`
	`if(t[index] == k)`
	`{`
	`position = index;`
	`}`
	`else {/* rien faire */ }`
	`index = index + 1;`
	`}`
	`// Retourner résultat`
	`return(position);`
	`}`
	`}`

Modèle algorithmique	Java

Exercice 6-16 - Trouvez la valeur maximale dans un tableau de taille N.

DONNÉES: T (réfère à un tableau de N valeurs)
N (nombres de valeurs dans le tableau)
INTERMÉDIAIRES: Index (index de T allant de 0 à N-1)
RÉSULTATS: Max (valeur maximale dans le tableau)
EN-TÊTE: Max ← MaxDansTableau(T, N)
MODULE

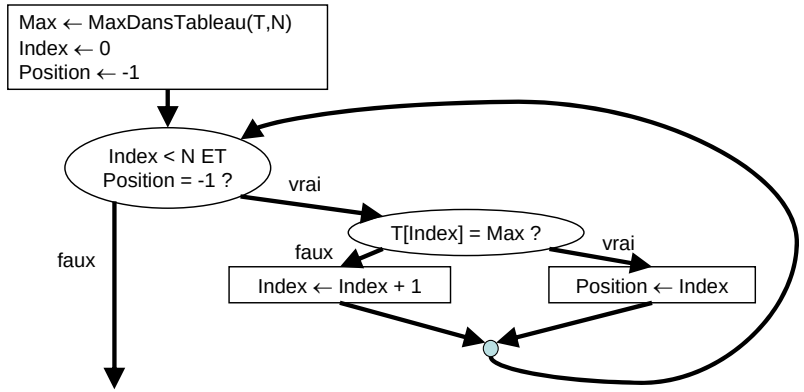



```
public static int maxDansTableau(int [] t)
{
   // DONNÉES: t – réfère à un tableau de int`s
   int n;  // La longueur du tableau
   // RÉSULTATS:
   int max; // Le maximum dans le tableau
   // INTERMÉDIAIRES:
   int index; //Index de t
   // Initialise n
   n = t.length;
   // Module de l'algorithme
   max = t[0];
   index = 1;
   while (index < n)
   {
      if (t[index] > max)
      {
         max = t[index]; //mise à jour
      }
      else
      {
         /* rien faire */ ;
      }
      index = index + 1;
   }

   // Retourner résultat
   return(max);
}
```

Exercice 6-17 (a) - Trouvez la position de la première occurrence de la valeur maximale d'un tableau de taille N.

DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
INTERMÉDIAIRES:
Index (index de T allant de 0 à N-1)
Max (valeur maximale dans le tableau)
RÉSULTATS: Position (Position de la première valeur maximale dans le tableau)
EN-TÊTE: Position ← MaxPosDansTableau(T,N)
MODULE:




```
public static int maxPosDansTableau(int [] t)
{
   // DONNÉES: t,  un tableau de int's, et n longueur
   int position; //RÉSULTAT: Position de max dans tabl.
   // INTERMÉDIAIRES:
   int index; //Index de t
   int max; // valeur maximale
   // Module
   max = maxDansTableau(t); //obtenir la valeur maximale
   index = 0; // recherche max dans tableau
   position = -1;
   while(index < t.length && position == -1)
   {
     if(t[index] == max)
       {  position = index; }
     else
       {  index = index + 1; }
   }
   return(position); // Retourner le résultat
}
```

Exercice 6-17 (b) - Trouvez la position de la première occurrence de la valeur maximale d'un tableau de taille N.

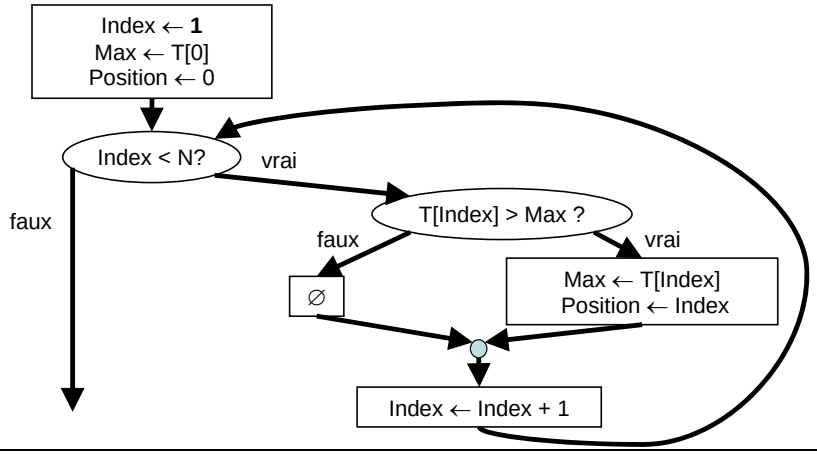
DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
INTERMÉDIAIRES:
Max (valeur maximale dans le tableau)
RÉSULTATS: Position (Position de la première valeur maximale dans le tableau)
EN-TÊTE: Position ← MaxPosDansTableau(T,N)
MODULE:
Max ← MaxDansTableau(T,N)
Position ← OùEstK(T,N,Max)

```
public static int maxPosDansTableau(int [] t)
{
   // DONNÉES: t, réfère à un tableau de int's
   int position; // RÉSULTAT: Position de max dans tabl.
   // INTERMÉDIAIRES:
   int n;// La longueur du tableau
   int index; //Index pour tableau
   int max; // valeur maximale
   // Module
   max = maxDansTableau(t); // obtenir valeur maximale
   position = oùEstK(t, max);  // trouve la position
   return(position); // Retourner le résultat
}
```

Modèle algorithmique	Java

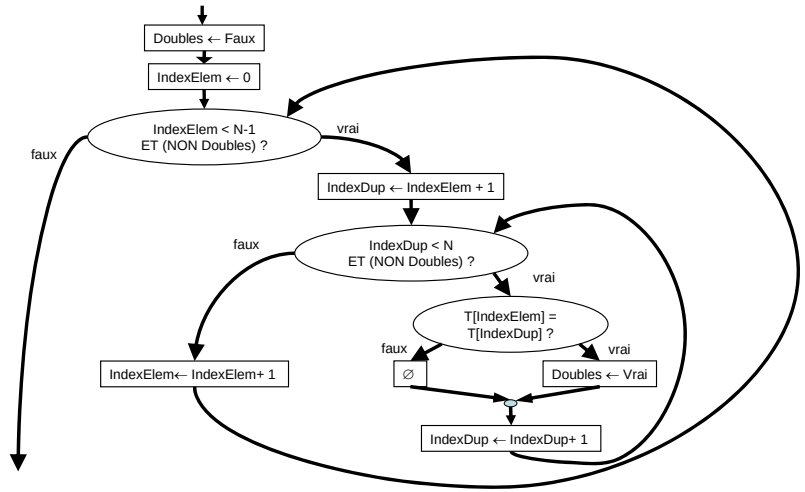
Exercice 6-17 (c) - Trouvez la position de la première occurrence de la valeur maximale d'un tableau de taille N.

DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
INTERMÉDIAIRES:
Index (index de T de1 à N)
Max (valeur maximale dans le tableau)
RÉSULTATS: Position (Position de la première valeur maximale dans le tableau)
EN-TÊTE: Position ← MaxPosDansTableau(T,N)
MODULE:




```
public static int maxPosDansTableau(int [] t)
{
   // DONNÉES: t, réfère à un tableau de int's
   int position; //RÉSULTATS: Position de max dans tabl.
   // INTERMÉDIAIRES:
   int index; //Index pour tableau
   int max; // valeur maximale
   // Module
   index = 1;
   max = t[0];
   position = 0;
   while(index < t.length)
   {
     if(t[index] > max)
     {
       max = t[index];
       position = index;
     }
     else index = index + 1;
   }
   return(position); // Retourner le résultat
}
```

Modèle algorithmique	Java
Exercice 6-18 - Vérifiez si un tableau de N valeurs contient ou non des valeurs dupliquées.	

DONNÉES: T (Réfère à un tableau de nombres)
N (Nombre d'éléments dans le tableau)
INTERMÉDIAIRES:
IndexElem (Index du tableau l'élément courant)
IndexDup (Index du tableau pour chercher un duplicat à l'élément courant)
RÉSULTATS: Doubles (Booléen: vrai si tableau a des doubles, sinon faux)
EN-TÊTE: Doubles ← AdesDoubles(T,N)
MODULE:
 
 	
	`public static boolean aDesDoubles(int [] t)`
	`{`
	`  // DONNÉES: t, réfère à un tableau de int's,`
	`  //  t.length utilisé pour n`
	`  boolean doubles; // RÉSULTATS: vrai si dupls trouvés`
	`  int indexElem; // INTMED: Index de l'élément courant`
	`  int indexDup; // INTMED: index pour chercher duplicats`
	`  // ALGORITHM MODULE`
	`  doubles = false;`
	`  indexElem = 0;`
	`  while (indexElem < t.length-1 && !doubles)`
	`  {`
	`  indexDup = indexElem + 1;`
	`  while(indexDup < t.length && !doubles)`
	`  {`
	`  if (t[indexElem] == t[indexDup])`
	`  {`
	`  doubles = true; // trouvé a duplicate`
	`  }`
	`  else { /*rien faire*/ }`
	`  indexDup = indexDup + 1;`
	`  }`
	`  indexElem = indexElem + 1;`
	`  }`
	`  return(doubles); // Retourner résultat`
	`}`

Exercice 6-19: Comparaisons de chaînes **`String`**

- Quelle est la valeur de **`résultat`** pour ces exemples?
  - Exemple 1:

```
String str1 = "abcde" ;
String str2 = "abcfg" ;
int résultat = str1.compareTo(str2);
```

`résultat` est plus petit que zéro

  - Exemple 2:

```
String str1 = "abcde" ;
String str2 = "ab" ;
int résultat = str1.compareTo(str2);
```

`résultat` est plus grand que zéro