INSA TOULOUSE

Domaine Mathématiques et Algorithmique
INSA 1ère année

Algorithmique 1

Sujets des Travaux Dirigés

D. Le Botlan
contact.lebotlan@insa-toulouse.fr

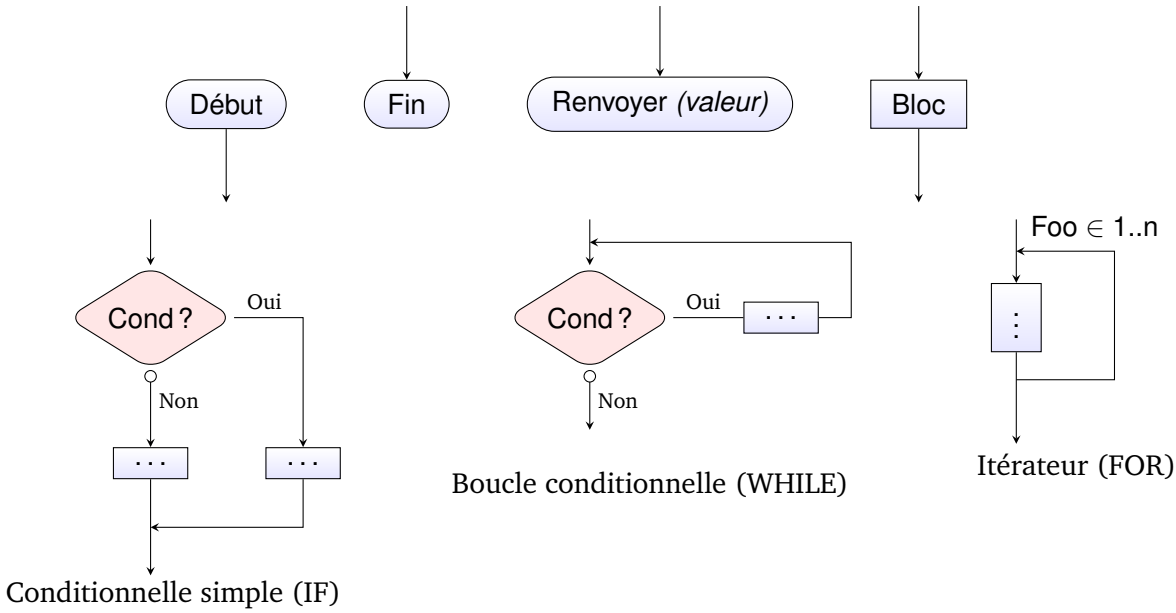
Mode d'emploi

Déroulement des séances

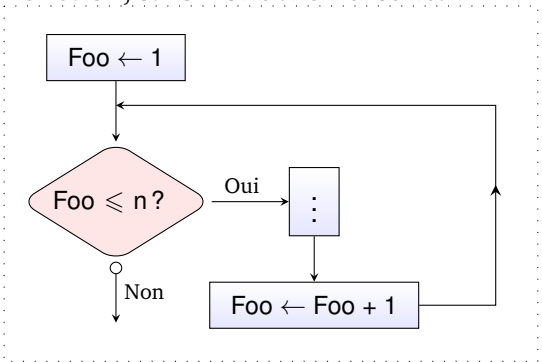
- Vous travaillez par équipes de quatre étudiants.
- Pour résoudre le problème posé, vous pouvez avoir besoin d'utiliser des concepts qui n'ont pas encore été vus en cours. C'est un choix pédagogique : feuilletez l'aide-mémoire Ada si besoin.
- Pendant la séance, vous pouvez poser des questions de compréhension à l'enseignant, mais il ne résoudra pas le problème à votre place.
- Aucun corrigé officiel des sujets ne vous sera remis (éventuellement un corrigé partiel vous permettant de vérifier votre propre travail). Par contre, votre encadrant se fera une joie d'examiner et de commenter ce que vous aurez produit. N'hésitez pas à le solliciter lorsque vous ne comprenez pas un point précis.

Algorigrammes

Voici les éléments utilisables dans les algorigrammes :

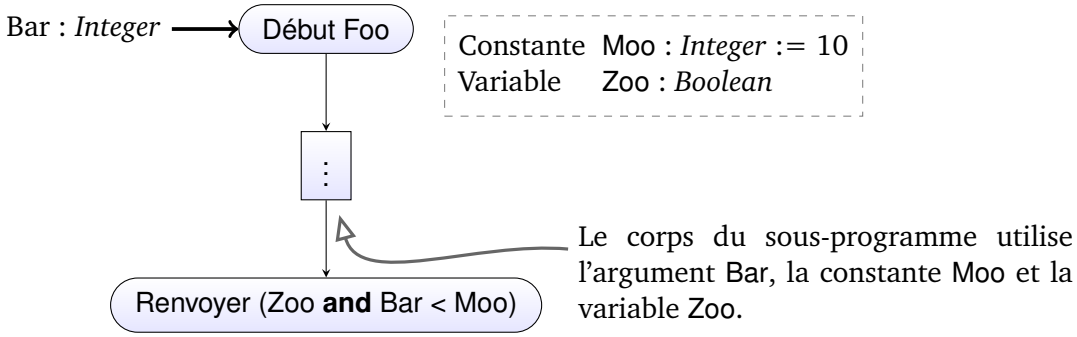


- Un `bloc` peut être un appel de sous-programme ou une affectation de variable, par exemple pour enregistrer dans la variable Bar le résultat renvoyé par la fonction Foo (la fonction Foo est définie ci-après) : `Bar ← Foo (18)` ou pour effectuer un calcul `Bar ← Bar × 2 − 1`
- Le bloc `Renvoyer` ne peut apparaître qu'à la fin d'une **fonction**, sans instruction ensuite.
- Le bloc **FOR** ci-dessus est équivalent à cette boucle **WHILE** :



Sous-programmes et variables locales

Voici un exemple de sous-programme Foo recevant un argument Bar de type entier, et dans lequel on définit une variable locale et une constante.



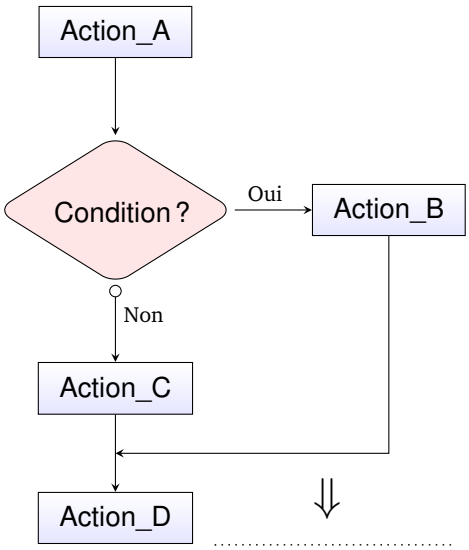
Distinction entre IF et WHILE

Lorsqu'un test conditionnel apparaît dans un algorigramme, il peut s'agir soit d'un **if**, soit d'un **while** (examiner la différence des deux structures sur la page précédente).

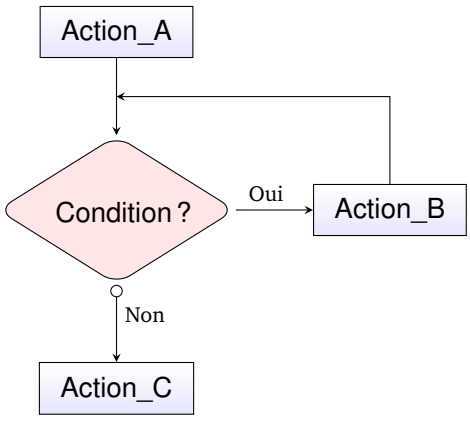
- Si les deux branches OUI/NON se rejoignent toujours, alors c'est un **if**.
- Si la branche OUI finit toujours par boucler juste au dessus du test, alors c'est un **while**, et de plus la branche NON ne doit jamais rejoindre la branche OUI.
- Tous les autres cas sont mal formés et ne pourront pas être traduits en Ada.

Traduction en Ada

Pour vous aider à traduire vos algorigrammes en Ada, voici quelques exemples de traduction :

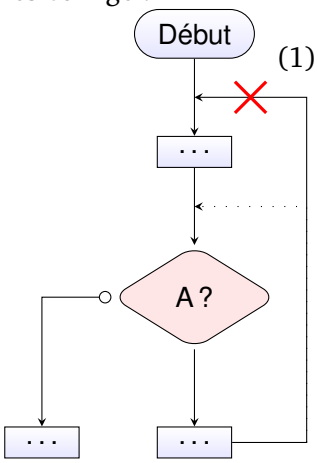


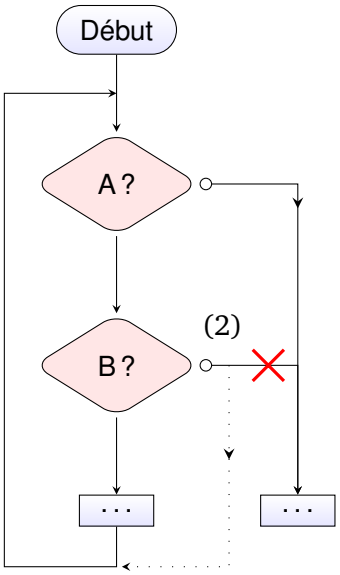

```
Action_A ;
if Condition then
   Action_B ;
else
   Action_C ;
end if ;
Action_D ;
```

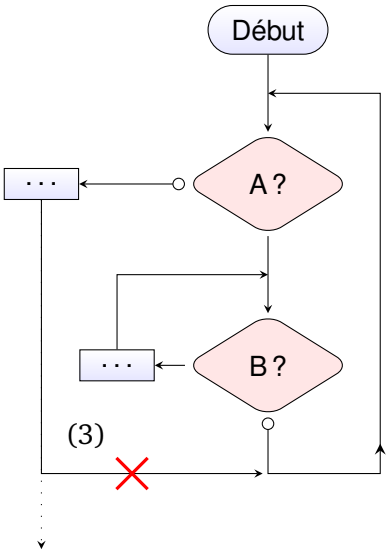



```
Action_A ;
while Condition loop
   Action_B ;
end loop ;
Action_C ;
```

Lorsqu'un algorigramme est mal formé, il n'est pas possible de le traduire directement en Ada. Les algorigrammes suivants sont mal formés. Les flèches en pointillés indiquent une manière possible de les corriger.







(1) Une boucle doit revenir juste au dessus de sa condition (ici A ?)

(2) Les deux branches d'une conditionnelle (ici B ?) doivent toujours se rejoindre.

(3) Les deux branches d'une boucle conditionnelle (ici A ?) ne doivent jamais se rejoindre.

Objectifs du TD

- Concevoir un algorithme simple utilisant plusieurs acteurs
- Manipuler des expressions numériques et booléennes
- Écrire des fonctions Ada
- Écrire un programme complet

Prérequis

- Notion d'acteur (cours)
- Notion de typage (exercices sur Moodle)

Note de Service

Service Recherche & Développement

De : Direction du Service R&D

Destinataires, pour exécution :
Équipe Qualité

Destinataires, pour information :
Production, DG

KakoTV

No friends ?™
Get a KakoTV !™

Date : 24 septembre

Objet : Mise en place d'un test de qualité en sortie de chaîne de fabrication

Contexte : Nos nouveaux téléviseurs 3D à effet de souffle (3D-ES) rencontrent un succès croissant, notamment depuis la première retransmission 3D-ES de *The Voice*. Cependant, les revendeurs ont attiré notre attention sur des défauts répétés de notre produit, les obligeant au mieux à remplacer les téléviseurs défectueux, et parfois à reconstruire certaines habitations détruites par un effet de souffle mal maîtrisé. Ceci engendre évidemment des coûts importants.

Après analyse par nos experts, les dysfonctionnements proviennent de défauts dans la fabrication de la cavité principale. Cette cavité servant de zone de résonance électromagnétique et sonore, ses dimensions doivent respecter une norme stricte, sans quoi l'effet 3D-ES échappe à tout contrôle.

Votre équipe est chargée de mettre en place un contrôle qualité automatique en sortie de la chaîne de production des cavités résonantes. Les cavités ne répondant pas aux normes seront recyclées.

Date d'échéance : 5 octobre

Spécifications : voir au dos.

Langage de programmation : Ada

KakoTV
Note de Service du 24 septembre

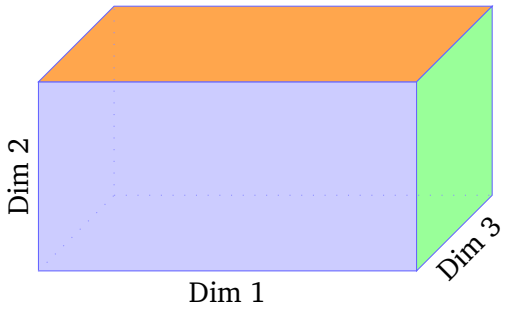
Spécification Nos cavités résonantes sont des parallépipèdes rectangles, appelés aussi « cuboïdes », voir le dessin ci-dessous.

Nous avons acheté des outils automatisés permettant de mesurer précisément la dimension des cavités résonantes. Ces outils sont pilotés par l'acteur Ada Outil_De_Mesure (spécification ci-dessous).

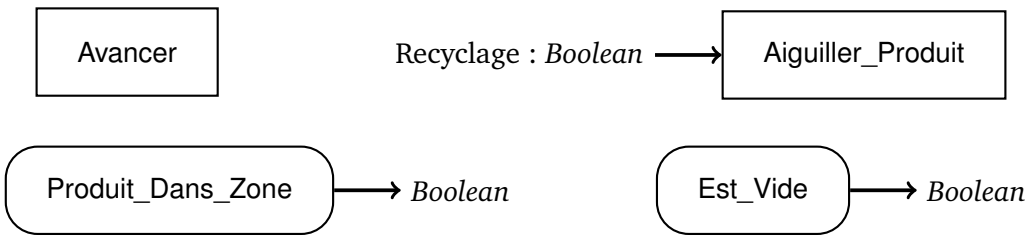
Un automate contrôlant le tapis roulant en sortie de chaîne de production est chargé de faire avancer les produits un par un. Une action de cet automate permet d'accepter le produit ou de le jeter pour recyclage.
Cet automate est piloté par l'acteur Ada Tapis (spécification ci-dessous).

Travail demandé Vous devez réaliser la programmation Ada du vérificateur de normes. Voir les consignes détaillées en annexe.

Cavité résonante :

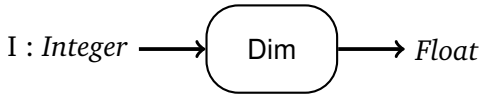


Actions pilotant le tapis



- Avancer : fait avancer le tapis de quelques centimètres.
- Produit_Dans_Zone : renvoie Vrai si un produit se trouve dans la zone de mesure (la zone de mesure fait partie du tapis).
- Aiguiller_Produit : dirige le produit (présent dans la zone de mesure) vers la sortie normale si l'argument Recyclage est Faux ou vers le recyclage si l'argument est Vrai.
- Est_Vide : renvoie Vrai s'il ne reste plus aucun produit à vérifier, Faux s'il reste des produits.

Action pilotant l'outil de mesure

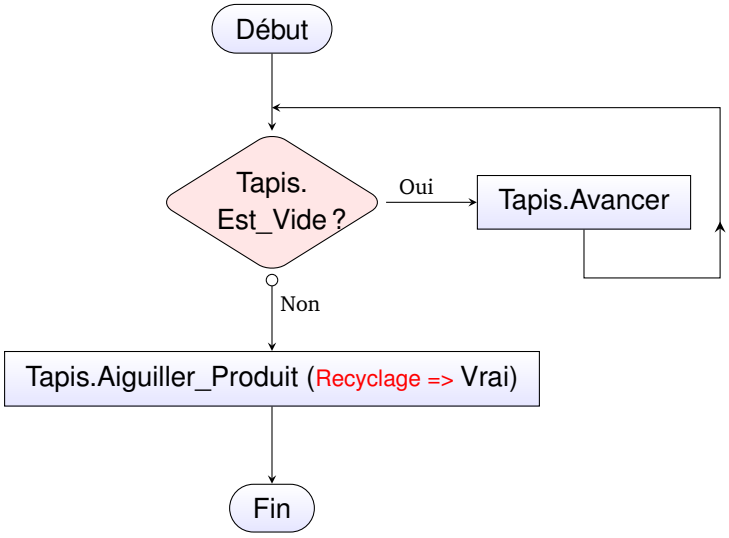


- Dim : renvoie la i-ème dimension du produit présent dans la zone de mesure, en mm (voir le schéma ; la 2ème dimension est la hauteur).

Annexe : Consignes détaillées

Première étape : pilotage du tapis

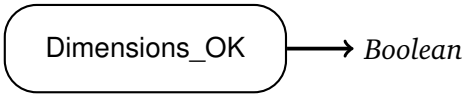
Un étudiant de 3ème année vous fournit un début d'algorithme, mais qui se révèle complètement faux. Pourquoi est-il incorrect ?



- ☞ Corriger l'algorithme pour que les produits défilent un par un dans la zone de mesure et soient **tous acceptés**.
- ☞ Le tapis doit arrêter d'avancer lorsqu'il n'y a plus de produits à tester.
- ☞ Réfléchissez encore : vous avez oublié de considérer au moins un cas de figure.

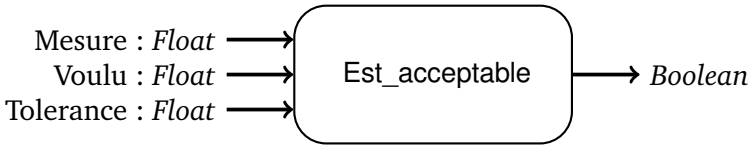
Deuxième étape : mesures des dimensions et comparaison avec la norme

L'objectif de cette étape est de fournir un algorithme pour la fonction suivante :



Hypothèse : un produit est présent dans la zone de mesure.

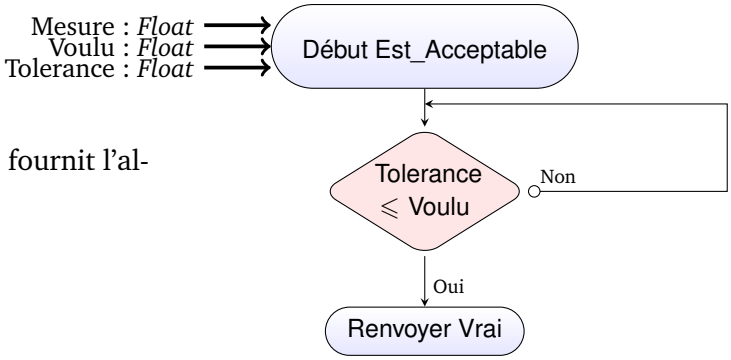
★ Pour faciliter l'écriture de cet algorithme, commencer par concevoir une autre fonction Est_acceptable qui prend trois arguments réels et qui renvoie un booléen :



- ○ Mesure, représentant la dimension mesurée (en mm)
- ○ Voulu, représentant la dimension souhaitée (en mm)
- ○ Tolerance, représentant la différence maximale acceptable entre Mesure et Voulu (en mm).

La fonction renvoie vrai lorsque la différence entre Mesure et Voulu est plus petite que Tolerance (en valeur absolue). Cette fonction **n'effectue pas de mesure**.

Votre douteux collègue de 3ème année vous fournit l'algorithme ci-contre :



★ Écrire maintenant l'algorithme de Dimensions_OK. Cette fonction **effectue les mesures** des dimensions du produit et renvoie **vrai** lorsque les trois dimensions sont correctes (à la tolérance près), et renvoie **faux** sinon. Trouvez une solution avec des IF et une variante sans IF.

Précision utile Les primates savent fabriquer des outils en bois ou en silex pour réaliser une tâche précise, mais ils les jettent ensuite, sans avoir l'idée de les garder (à la différence d'*homo-habilis*).

Les valeurs à respecter sont les suivantes :

Norme_Longueur	852.4 mm	(dimension 1)
Norme_Hauteur	600 mm	(dimension 2)
Norme_Largeur	23.7 mm	(dimension 3)
Tolerance	0.18 mm	

Troisième étape : algorithme complet

Complétez l'algorithme de pilotage du tapis pour qu'il accepte les bons produits et rejette les mauvais.

Quatrième étape : traduction en Ada

Traduire l'ensemble de votre algorithme en Ada. Voici les acteurs Ada disponibles :

Acteur Tapis *(automate du tapis roulant)*

```
package Tapis is

   -- Fait avancer le tapis de quelques centimètres
   procedure Avancer ;

   -- Renvoie Vrai si un produit se trouve dans la zone de mesure
   function Produit_Dans_Zone return Boolean ;

   -- Dirige le produit (présent dans la zone de mesure) vers la sortie normale si Recyclage est Faux
   -- ou vers le recyclage si Recyclage est Vrai
   procedure Aiguiller_Produit (Recyclage : Boolean) ;

   -- Renvoie Vrai s'il ne reste aucun produit à vérifier, Faux s'il reste des produits
   function Est_Vide return Boolean ;

end Tapis ;
```

Acteur Outil_De_Mesure

```
package Outil_De_Mesure is

   -- Renvoie la i-ème dimension du produit sur le tapis, en mm
   -- I=1 correspond à la longueur, I=2 à la hauteur, I=3 à la largeur
   function Dim (I : Integer) return Float ;

end Outil_De_Mesure ;
```

Cinquième étape : anticipation d'un risque

Afin de prévenir un éventuel dysfonctionnement des capteurs qui aboutirait au rejet de tous les produits du tapis, on nous demande de compléter l'algorithme pour que le tapis s'arrête lorsqu'ont été détectés 5 produits défectueux **d'affilée**.

Sixième étape (facultatif, mais formateur) : statistiques

Calculer dans une variable la longueur de la plus longue série de produits sans défaut d'affilée.

TD – Sujet 2

Objectifs du TD

- Comprendre la sémantique précise des instructions Ada
- Étudier un algorithme standard de calcul d'extremum

Prérequis

- Avoir compris les variables
- Connaître les différents blocs

Contexte

Notre SSII (société de services en ingénierie informatique) *BugBusters* a été contactée par la banque *Société Géniale* pour étudier un problème de virus informatique.
En effet, cette banque a découvert que le bilan de ses comptes n'est plus équilibré : de l'argent disparaît de certains comptes sans profiter à personne. Le responsable semble être un virus qui se propage discrètement sur leur réseau depuis plusieurs jours. Le directeur de la banque, très inquiet, nous a confié comme mission de comprendre précisément ce que fait ce virus et d'équilibrer les comptes.

Attention, toute l'affaire doit rester absolument secrète.

Mission

La banque a transmis un programme infecté à notre société. Des experts en virologie en ont extrait le virus, puis retiré le code de réplication et de propagation (qui ne nous intéressent pas ici). Il ne reste que la partie opérationnelle du virus que nous devons analyser.

☞ **Votre mission est d'analyser le virus et de comprendre ce qu'il fait.**

Ressources

Le code du virus a été traduit en langage d'assemblage, communément appelé **assembleur**.

L'assembleur Ce langage rudimentaire correspond exactement aux instructions exécutables par le processeur de l'ordinateur.
Pour information, le compilateur Ada, lorsqu'il compile un programme Ada, produit du code assembleur qu'il place dans un exécutable (dans `mission-exe`).
Une documentation succincte sur l'assembleur se trouve en annexe.

Travail demandé

1. Examinez le code du virus (page ci-après) et comprenez ce qu'il fait. Une manière efficace de procéder consiste à simuler le programme en écrivant sa *trace* (voir en Annexe).
2. Représentez l'algorithme sous forme d'algorigramme, le plus compact possible (c.-à-d., en simplifiant les opérations intermédiaires lorsque c'est possible), et en choisissant le nom des variables avec soin.
3. On suppose que les actions `lire_nombre_de_comptes`, `lire_solde_compte`, et `ecrire_solde_compte` sont définies dans l'acteur `Banque`. Écrivez le fichier .ads de l'acteur Banque.
4. Traduisez l'algorithme en Ada lisible.
5. Répondez aux questions suivantes :
   - Quelle différence entre EXTERN en assembleur et WITH en Ada ?
   - On suppose que chaque instruction assembleur s'exécute en un TIC d'horloge (l'horloge d'un processeur tourne à 1GHz environ). Combien de temps faut-il pour exécuter complètement le virus ? (en fonction de N, nombre de comptes, égal à 10^6 pour l'application numérique)
   - Que se passe-t-il si l'on retire l'instruction `add R8, 1` ligne 16 ?
6. Modifier le virus pour que l'euro qui a été retiré sur un compte soit crédité sur le compte du client le plus pauvre. Les comptes seront ainsi équilibrés.

Code opérationnel du virus

```
1  EXTERN lire_nombre_de_comptes
2  EXTERN lire_solde_compte
3  EXTERN ecrire_solde_compte

4  Code1 SECTION CODE
5          start:  mov R1,0
6                  call lire_nombre_de_comptes  ; Résultat dans R9
7                  mov R2, R9
8                  mov R8,1
9          loop1:  cmp R8, R2
10                 jmp GT, exit1
11                 call lire_solde_compte   ; Numéro de compte dans R8, résultat dans R9
12                 cmp R9, R1
13                 jmp LE, test1
14                 mov R1, R9
15                 mov R3, R8
16         test1:  add R8, 1
17                 jmp loop1
18         exit1:  mov R8, R3
19                 mov R9, R1
20                 add R9, -1
21                 call ecrire_solde_compte   ; Numéro de compte dans R8, nouveau solde dans R9
22 Code1 ENDS
```

Annexe : documentation assembleur succincte

Le langage assembleur utilise des variables particulières situées directement dans le processeur (et non pas en mémoire) : les *registres*. Ceux-ci sont numérotés : R1, R2, R3, ...
Un registre se comporte essentiellement comme une variable.

- **EXTERN** sert à importer une procédure ou fonction, un peu comme **WITH** en Ada.
- **SECTION**...**ENDS** sont des délimiteurs, de la même manière que **begin**...**end** en Ada.
- foo: Un mot suivi de deux-points sert à repérer un endroit dans le code, c'est une *étiquette*.
- **mov** x, y met la valeur de y dans le registre x
- **add** x, y ajoute le valeur de y au registre x
- **CALL** invoque une procédure ou fonction. Les paramètres éventuels sont placés dans les registres R8, R9, etc. Une fois l'appel terminé, le résultat éventuel (si c'est une fonction) est mis dans R9.
- **cmp** x, y compare les deux nombres x et y, c.-à-d., regarder si x est plus petit, égal, ou plus grand que y. Le résultat est mémorisé pour la suite.
- **jmp LE**, foo indique qu'il faut aller à l'instruction située après l'étiquette foo: seulement si la dernière comparaison renvoyait le résultat « plus petit que » (*Less or Equal*). Sinon, le programme continue à l'instruction suivante. La condition **GT** signifie *Greater Than*.

Question facultative : à chaque compte correspond un numéro de client (accessible via la fonction No_Client (No_Compte : *Integer*) **return** *Integer*). Trouver le client le plus riche, en prenant en considération tous ses comptes.

Trace d'un programme

Écrire la trace d'un programme consiste à écrire la valeur de toutes ses variables au fur et à mesure de son exécution. Quand une valeur est inconnue, on écrit "?" .

Voici le début de la trace du programme du virus, à compléter :

- ☞ Les numéros de ligne ne se suivent pas forcément : ils peuvent changer avec l'instruction **jmp**.
- ☞ Vous pouvez supposer qu'il y a 3 comptes dont les soldes sont 830 €, 141 000 € et 1680 €.

N'écrivez pas ici : recopiez ce tableau si vous voulez que ce poly puisse resservir.

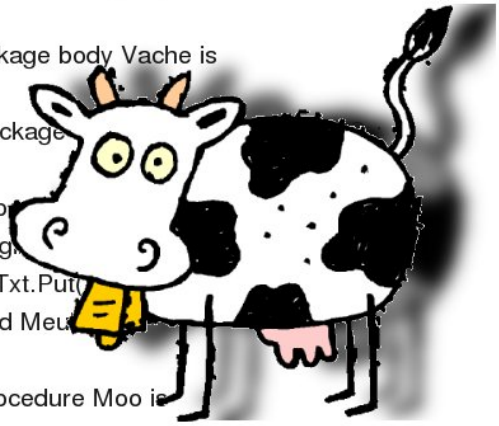
LIGNE	R1	R2	R3	R8	R9
5	0	?	?	?	?
6	0	?	?	?	3
7	0	3	?	?	3
8	0	3	?	1	3
...	...	...	...	...	...

(Prévoyez une vingtaine de lignes.)

```
with Ada.Text_IO ;

package body Vache is

  package
```



```
  pro
  beg
    Txt.Put(
  end Meu

  procedure Moo is
  begin
    Txt.Put("Moo !") ;
  end Moo ;

end Vache ;
```

Version du 1er juillet 2024.