Le langage Init

Licence Info S5
documentation COMPIL –

FIL

INIT est un tout petit langage impératif donné à titre d'illustration, dont les instructions sont des initialisations. On utilisera deux versions, l'une avec des types simples et l'autre étendue avec un type liste.

1 Version simple

Un programme INIT déclare des variables entières, puis les initialise soit par lecture au clavier, soit par affectation d'une constante entière. Une variable ne peut être initialisée que si elle a été déclarée, et n'est initialisée qu'une fois. Les identificateurs d'INIT sont ceux de Java (une lettre ou _ suivie éventuellement d'une suite de lettres, chiffres ou _). Les entiers sont des suites de chiffres.

Structure d'un programme

- Un *programme* est constitué d'un en-tête obligatoire, suivi d'une partie déclaration puis d'une partie instruction optionnelles.
- Un *en-tête* commence par le mot-clé `program`, suivi d'un identificateur, suivi d'un point-virgule (`;`).
- La partie déclaration consiste en une *liste de déclarations*. Une *déclaration* déclare une ou plusieurs variables de type entier, et commence par le mot-clé `int`. Suit ensuite une liste d'identificateurs de variable séparés par des virgules (`,`). Une déclaration termine par un point-virgule.
- La partie instructions consiste en une *liste d'instructions*. Une *instruction* est soit une affectation, soit une lecture, et termine par un point-virgule.
- Une *affectation* affecte à une variable de type entier une constante entière. Elle consiste en un identificateur de variable suivi des caractères deux-points et égal (`:=`) suivis d'un entier.
- Une *lecture* commence par le mot-clé `read` suivi d'un identificateur de variable de type entier.

Exemples de programmes corrects De par la syntaxe du langage on peut écrire un programme INIT qui ne semble pas avoir grand sens (par exemple qui ne déclare pas de variables) : INIT est un langage à but uniquement pédagogique !

Un programme minimal :

```
program monprog ;
```

Un programme qui déclare une variable :

```
program     monprog2;

int x;
```

Un programme qui déclare des variables et les initialise :

```
program mon_prog3;
int x,y;
int z;
read z;
y := 3;
x := 345;
```

Exemples de programmes incorrects Un programme qui contient des unités lexicales non autorisées (comme `write` par exemple) ou ne respecte pas la grammaire du langage (par exemple entrelace déclarations et initialisations) est bien sûr incorrect. Par ailleurs un programme lexicalement et syntaxiquement correct peut être sémantiquement incorrect, par exemple :


```
program bourde1;
int x;
read y;
```

ou

```
program bourde2;
int x;
x := 3; read x;
```

2 Version étendue

On étend maintenant INIT avec un nouveau type de données : les *listes d'entiers*. On peut donc déclarer des variables de type liste (d'entiers implicitement). Ces variables ne peuvent être initialisées par une opération de lecture : elles sont uniquement affectées. On représente une liste d'entiers par une liste éventuellement vide d'éléments encadrée par `[` et `]`. Un élément peut être soit un entier, soit une liste.

Structure d'un programme Elle reste globalement la même que dans la version simple.

- Une *déclaration* déclare soit des variables de type entier (auquel cas elle commence par le mot-clé `int`), soit des variables de type liste (auquel cas elle commence par le mot-clé `list`). Le mot-clé est suivi comme auparavant par une liste de variables.
- Une *affectation* affecte à une variable de type entier une constante entière ou à une variable de type liste une liste d'entiers.
- Une *lecture* commence par le mot-clé `read` suivi d'un identificateur de variable de type entier.

Exemple de programme correct

Un programme qui utilise des listes :

```
program progliste;
list x,y,z;
y := [ [3] [2 1] 1];
x := [ [ [1] 3 ] 5 ];
z := [];
```

Le langage Ava est un langage de programmation[1] impératif simplissime[2] : il ne contient pas de fonctions ni procédures, ne possède que des variables globales de type entier ou booléen, et n'a pas la notion de portée. À la fin du module de COMPIL, vous devriez avoir réalisé un tout petit compilateur pour Ava. Ce document présente uniquement les caractéristiques syntaxiques et sémantiques du langage.

Variables, types et expressions

Les *variables* sont désignées par des identificateurs et doivent être déclarées. Les déclarations de variables (entières et booléennes respectivement) sont de la forme :

```
int var1, var2, ... , varn ;
boolean var1, var2, ... , varn ;
```

Les déclarations des deux types peuvent être entrelacées. Une variable ne peut être déclarée qu'une fois et est implicitement initialisée à la déclaration (à 0 pour une variable entière et faux pour une variable booléenne). Les identificateurs ont la même syntaxe que les identificateurs Java.

Une *expression* Ava est de type entier ou booléen. Une *expression entière* est une expression arithmétique parenthésée habituelle, à résultat entier (combinaison des opérateurs d'addition `+`, soustraction `-`, multiplication `*`, division `/`, opposé `-`, modulo `mod`). Le moins unaire est prioritaire sur l'opérateur `mod`, lui-même prioritaire sur `*` et `/`, eux-même prioritaires sur `+` et `-`. Une expression est construite uniquement au moyen de variables déclarées de type entier et de constantes de type entier (par ex. `52`).

Une expression booléenne est construite à partir des opérateurs classiques (`and`, `or`, `not`), de parenthèses, de variables déclarées de type booléen, des constantes `true` et `false`, et de comparaisons entre expressions entières par les opérateurs classiques (égalité `=`, différence `/=`, `<=`, `<`, `>`, `>=`). L'opérateur `not` est prioritaire sur `and` qui est prioritaire sur `or`. Les opérateurs arithmétiques sont prioritaires sur les opérateurs de comparaison qui sont prioritaires sur les opérateurs booléens.

Instructions

Ava propose les instructions impératives suivantes dont l'exécution est standard.

Une affectation est de la forme `var := expr ;` où `expr` est une expression Ava. Toute variable qui est le membre gauche d'une affectation doit avoir été déclarée, de type entier ou booléen. La concordance des types doit être respectée : `var` et `expr` doivent être toutes les deux de type entier ou toutes les deux de type booléen. On aura par exemple :

```
int x,y; boolean b;
x := 3; y := 6; b := x<(y+1);
```

mais pas `x := b ;`.

Une impression est de la forme :

```
write ( %i , exprEnt ) ; write ( %s, chaine ) ; write ( %b , exprBool) ;
writeln ( %i , exprEnt ) ; writeln ( %s, chaine ) ; writeln ( %b , exprBool) ; writeln ;
```

où `exprEnt` désigne une expression Ava entière, `exprBool` désigne une expression Ava booléenne, `var` désigne un identificateur de type booléen, et `chaine` est une chaîne de caractères à la Java (ou à la C, du type `"coucou"`). `%i`, `%b` et `%s` sont des formats d'impression correspondant respectivement aux formats entier, booléen et chaîne. Comme en Java les guillemets despécialisés à l'intérieur d'une chaîne sont représentés par `\"` et le retour à la ligne est représenté par `\n` (par exemple `"il dit \"coucou\"\n"`). L'instruction `writeln` déclenche un retour à la ligne, mais pas `write`. On aura par exemple :

```
write(%s,"b vaut "); write(%b,b); write(%s," et x+2 vaut "); write(%i,x+2); writeln;
```

déclenchant l'affichage de `> b vaut vrai et x+2 vaut 5`.

[1] Ava v1 a été introduit par Yves Roos dans ce cours de compilation. La version 2 ajoute le type booléen et modifie la syntaxe des impressions.

[2] Donc pas pratique du tout pour programmer !

Une lecture est de la forme `read var ;` où `var` est une variable préalablement déclarée de type entier uniquement.

Les conditionnelles sont de la forme :

```
if exprBool  then listeInstruction end if ;
if exprBool then listeInstruction else listeInstruction end if ;
```

dans lesquelles `listeInstruction` désigne une séquence d'instructions, et `epxrBool` est une expression de type booléen. Le mot `end` et le mot `if` peuvent être séparés par un nombre quelconque de blancs et/ou retour à la ligne.

Une itération est de la forme

```
while exprBool loop listeInstruction end loop ;
```

avec les mêmes contraintes que pour les conditionnelles.

Structure d'un programme

Un programme commence par le mot-clé `program` suivi d'un littéral de type chaîne (le nom du programme, qui ne doit plus être utilisé dans la suite du code) et d'un `;`. Ensuite viennent une suite facultative de déclarations, puis une suite facultative d'instructions. Comme en Eiffel un *commentaire* est indiqué par `--` (et s'étend du `--` jusqu'à la fin de la ligne). Un commentaire peut apparaître à n'importe quel emplacement du programme. Les espacements et retours à la ligne ne sont pas significatifs. On pourra ainsi écrire :

```
program
        "fact"
;
```

Exemple

Voici un calcul de factorielle en Ava.

```
 -- calcul de factorielle
program "Fact" ;
  int x, xbis ;
  boolean fini;
  int res;
  write (%s,"entrer un entier positif\n");
  read x; xbis := x; -- mém de x
  res := 1; -- résultat
  fini := x = 1;
  if x /= 0
  then
    while not fini
    loop
       res := res * x;
       x := x - 1 ;
       fini := x = 1;
    end loop ;
  end if ;
  write(%s,"factorielle("); write(%i,xbis);
  write(%s,")="); write(%i,res);
  writeln ;
```

Voici un calcul de pgcd en Ava.

```
program "pgcd" ;
-- calcul de pgcd
int x , y , x2 , y2  ;
writeln (%s,"Entrer un entier : ");
read x; x2 := x ;
writeln (%s, "Entrer un entier : ");
read y  ; y2 := y ;
if x > 0 and y > 0 then
  while x /= y loop
    if x > y then
      x := x - y ;
    else
      y := y - x ;
    end if ;
  end     loop ;
  write (%s,"Le pgcd de "); write(%i,x2);
  write (%s, " et de "); write(%i, y2 );
  write(%s, " est "); write(%i,x); writeln;
else
  writeln (%s,"Les entiers doivent etre strict
end if ;
```