Le langage JF2
version 2013-a
manuel de référence

Michel Billaud

2 janvier 2013

Le langage JF2

Le système JF2 (Just For Fun) est un interprète pour le langage de programmation JF2. Plus précisément, c'est un programme écrit en Python qui lit un *fichier source* écrit dans un langage de programmation intermédiaire entre les langages d'assemblage et Fortran 0, le traduit en code intermédiaire qui est ensuite exécuté.

Ce langage permet d'exprimer des algorithmes simples, exprimés à l'aide d'expressions arithmétiques et de branchements conditionnels ou inconditionnels. Ces algorithmes portent sur des variables (*scalaires*) ou des *tableaux* (qui peuvent être multidimensionnels) de type entier.

Certaines instructions permettent de faire les entrées-sorties nécessaires.

Les opérations arithmétiques sont limitées aux nombres entiers.

Ce manuel est un vague pastiche de `http://www.fortran.com/FortranForTheIBM704.pdf`

Table des matières

1 Propriétés générales d'un programme JF2

Un programme JF2 est composé d'une suite de lignes, contenant divers types d'*instructions* et de *déclarations* qui seront détaillées plus loin.

1.1 Un exemple de programme

Le bref exemple de la figure 1 permet de montrer l'allure générale et quelques propriétés d'un programme JF2.

```
# écrit les carrés des                                  1
# entiers de 1 à 10                                     2
    declare i                                           3
    i = 1                                               4
boucle                                                  5
    println  i,"->",i*i                                 6
    i = i + 1                                           7
    jump boucle if i <= 10                              8
                                                        9
    stop                                                10
```

FIGURE 1 – Un exemple de programme

1.2 Fonctionnement du programme

Le programme place (ligne 4) dans la *variable* `i` la valeur constante `1`, puis (6) affiche la valeur de cette variable, une flèche, et le carré de la variable sur une ligne du terminal de l'utilisateur. La valeur de `i` est ensuite augmentée d'une unité (7) et l'exécution reprend en `boucle`[1] au niveau de l'affichage, tant que (8) la valeur de `i` est restée inférieure ou égale `10`. Enfin, quand `i` dépasse `10`, l'exécution du programme se termine.

1.3 Présentation du programme

- Chaque instruction ou déclaration est tapée sur une ligne.
- Le caractère dièse (#) commence un *commentaire*, qui se poursuit jusqu'à la fin de la ligne.
- r Les *étiquettes*, comme `boucle` ici, servent à repérer des positions dans le code du programme. Elles sont *déclarées* en les faisant figurer en début de ligne.
- Il est possible de combiner une déclaration d'étiquette, une instruction (ou déclaration) ainsi qu'un commentaire sur la même ligne. Dans ce cas l'éti-

1. c'est le cas de le dire

quette doit être séparée de l'instruction proprement dite par un ou plusieurs espaces. Exemple :

```
boucle   println i,"->",i*i # affichage
```

2 Constantes, variables et indices

Tout langage de programmation impérative permet la représentation de constantes numériques et de quantités variables. JFF permet aussi une notation indicée pour des tableaux de variables.

2.1 Constantes

JFF utilise les nombres entiers en virgule fixe, de taille illimitée.

2.2 Variables

Les variables sont identifiées par un nom (*identificateur*), est composé d'au moins une lettre, suivie par un nombre quelconque de lettres et de chiffres.

Exemple d'identificateurs corrects : `somme`, `p2x437`. Identificateurs invalides : `b_52`, `3mustafa3`.

Les variables sont obligatoirement déclarées par `declare` préalablement à toute utilisation dans la suite du programme.

2.3 Tableaux et indices

Les variables peuvent être groupées en *tableaux* à une ou plusieurs `dimensions`. Par exemple la ligne

```
declare  v(10), mat(2,3)
```

déclare un tableau uni-dimensionnel (aussi appelé *vecteur*) de 10 variables `v(1)`, `v(2)` ... `v(10)`, ainsi qu'une matrice à 2 lignes et 3 colonnes dont les éléments `mat(`i`,`j`)` sont repérés par des *indices* entiers tels que $1 \leq i \leq 2$ et $1 \leq j \leq 3$.

Dans la suite, le terme *variable scalaire* désignera les variables qui ne sont pas des éléments de tableaux (*variables indicées*).

3 Expressions

Les *expressions* sont formées de constantes, d'identificateurs, de parenthèses, de virgules et d'opérateurs combinées selon des règles précises pour former une expression mathématique qui a un sens.

3.1 Syntaxe

- toute constante ou variable (scalaire ou indicée) est une expression ;
- si E_1 et E_2 sont des expressions, E_1 op E_2 est une expression si op désigne une des opérations arithmétiques+ (somme), - (différence), * (produit), / (quotient), % (modulo).
- si E est une expression, (E) et -E sont des expressions.

3.2 Priorités

Lorsque la hiérarchie des opérations n'est pas précisée complètement par des parenthèses, elles doivent être comprises dans l'ordre suivant

- opérations multiplicatives (* / %) d'abord,
- opérations additives (+ -) ensuite,

avec groupement à partir de la gauche. Par exemple

```
a / b * c  +  d - e - f
```

doit être lu

```
((a / b) * c)  +  ((d - e) - f)
```

Note. Dans une expression peuvent apparaître des variables indicées, dont les indices sont eux-mêmes des expressions.

Exemple : `3*mat(i+1,j-1) + 42`

Les indices doivent correspondre en nombre à la déclaration du tableau.

4 Affectation

L'affectation transfère la valeur d'une expression dans une variable, scalaire ou indicée. Exemples

```
declare v(10), m(3,3), i
...
i           = 12
v(3)        = i + 1
v(m(i-1,2)) = m(v(i),1) + 42
```

5 Contrôle

5.1 Arrêt

L'instruction

```
stop
```

met fin à l'exécution du programme.

5.2 Saut

L'instruction

`jump` *étiquette*

provoque le transfert du contrôle à l'endroit indiqué par l'étiquette.

5.3 Saut conditionnel

L'instruction

`jump` *étiquette* `if` E_1 *comparaison* E_2

provoque le transfert du contrôle à l'endroit indiqué par l'étiquette si la comparaison des valeurs de E_1 et E_2 fournit le résultat attendu. Sinon l'exécution se poursuit à l'instruction suivante.

Les comparaisons disponibles sont : < (inférieur strictement), <= (inférieur ou égal), > (supérieur strictement), >= (supérieur ou égal), == (égalité), != (inégalité).

Exemple

```
jump suite if v(i)!=0
```

5.4 Appel de sous-programe

L'instruction

`call` *étiquette*

provoque le transfert du contrôle à l'endroit indiqué par l'étiquette, et le rangement de l'adresse de l'instruction suivant sur la pile des adresses de retour, pour être utilisée ultérieurement par `return`.

5.5 Retour de sous-programe

L'instruction

`return`

provoque le transfert du contrôle à l'adresse retirée du sommet de la pile des adresses de retour.

Si la pile est vide, le comportement est indéfini.

```
    call bonjour
    call aurevoir
    stop

bonjour
    print "bonjour"
    return
```

```
aurevoir
    print "aurevoir"
    return
```

6 Entrées-sorties

6.1 Affichage

Les instructions `print` et `println` font afficher une suite de valeurs d'expressions et d'éléments de texte (chaînes de caractères entre guillemets).

Exemple

```
# affiche n premières cases
# du tableau t
    println n, "élements :"
    i = 1
boucle
    jump fin if i > n
    print "t(",i,")=",t(i)
    i = i + 1
    jump boucle
fin
    println   # saut de ligne
```

Dans le programme les divers paramètres (chaines ou expressions) d'un `print` ou `println` sont séparés par des virgules. Lors de l'affichage ils sont représentés séparés par des points.

Deux différences entre `print` et `println`

– `println` provoque un saut de ligne après l'affichage des éléments

– `print` a obligatoirement au moins un paramètre/

6.2 Lecture

L'instruction `input` provoque la lecture d'une ligne contenant une suite de valeurs à affecter à une liste de variables.

Exemple

```
    declare a, b, max
    print "donnez a et b : "
    input a, b

    max = a
    jump affichage if b <= a
    max = b
affichage
    println "le plus grand est", max
```

```
stop
```

7 Exemples de programmes

7.1 Somme des carrés

```
# calculer la somme des carrés
# des entiers de 1 à n
    declare n
    print "Valeur de n = "
    input n

    declare i, somme
    somme = 0
    i = 1
boucle
    somme = somme + i*i
    i = i + 1
    jump boucle if i <= n

    println "somme = ", somme
    stop
```

7.2 Tours de Hanoï

```
#
# tours de hanoi
#
    declare from(10), to(10), nb(10)
    declare top
    declare f, t, b, n

    print "nombre de tours = "
    input n

    from(1) = 1  # tour de départ
    to(1)   = 2  # tour d'arrivée
    nb(1)   = n
    top     = 1  # sommet de pile

    print "*** tours de hanoi avec"
    println   n,"tours."

loop
```

```
    f = from(top)                                    20
    t = to(top)                                      21
    b = 6 - (f+t)  # la 3ieme tour                   22
    n = nb(top)                                      23
    top = top - 1                                    24
                                                     25
    jump general if n > 1                            26
                                                     27
    println f,"->",t                                 28
    jump next                                        29
general                                              30
    from(top+3) = f                                  31
    to  (top+3) = b                                  32
    nb  (top+3) = n-1                                33
                                                     34
    from(top+2) = f                                  35
    to  (top+2) = t                                  36
    nb  (top+2) = 1                                  37
                                                     38
    from(top+1) = b                                  39
    to  (top+1) = t                                  40
    nb  (top+1) = n-1                                41
                                                     42
    top = top+3                                      43
next                                                 44
    jump loop if top > 0                             45
    stop                                             46
```

Index