Licence 3 Informatique
Module : théorie des langages et compilation

Projet
Turtl

L'objectif de ce projet est de créer un interpréteur pour un langage permettant de déplacer une ou plusieurs tortues dans un jardin. Les spécificités du langage sont détaillées dans la suite du sujet.

Le projet peut se faire seul ou en binôme et est à rendre au plus tard le jeudi 13 décembre par email à fabien.garreau@univ-angers.fr, theo.lecalvar@univ-angers.fr et sara.tari@univ-angers.fr.

1 Généralités

Le projet est divisé en deux parties : la reconnaissance du langage et l'interprétation du langage.

La reconnaissance du langage sera effectuée dans un premier temps. Les explications nécessaires à l'interprétation du langage (par exemple l'interface graphique) seront fournis le jeudi 29 novembre lors d'une mise à jour du sujet.

Afin de tester votre projet, vous pouvez récuperer les exemples disponibles sur https://gdsn.fr/compil/projet_1819/exemples.

Vous pouvez récupérer la base bison utilisé lors des TP sur la page https://gdsn.fr/compil/.

2 Langage

2.1 Instructions simples

La tortue peut être déplacée en utilisant des instructions (Exemple 1). Il n'est possible d'utiliser qu'une seule instruction par ligne. Une tortue peut :

- avancer d'une case avec l'instruction `avance [nombre]`
- reculer d'une case avec l'instruction `recule [nombre]`
- sauter (avancer de deux cases en sautant par dessus une case occupée) avec l'instruction `saute [nombre]`
- tourner d'un quart de cercle avec l'instruction `tourne` <`à droite`|`à gauche`> [1]

1. Dans le sujet on suivra le formalisme bash, les paramètres obligatoires sont notés entre < et > et les paramètres facultatifs entre [et].

Les instructions ont un argument facultatif correspondant au nombre de fois où effectuer une instruction. Ce nombre peut être exprimé par une constante entière ou une expression arithmétique utilisant les opérateurs suivant $+$, $-$, $*$, $/$ et $()$. Si le résultat n'est pas un entier on arrondira à l'entier le plus proche. Le nombre de répétitions est placé après l'instruction et peut être suivi du mot `fois`. Ainsi, les instructions suivantes permettent toutes d'avancer la tortue de 2 cases :

— `avance 2 fois`
— `avance 2`
— `avance (3*3+1)/5 fois`

Lorsqu'il existe plus d'une tortue, les instructions s'appliquent à toutes les tortues. Il est cependant possible d'affecter des actions à une tortue donnée en ajoutant `@n` à la fin d'une instruction, où n correspond au numéro de la tortue ciblée (Exemple 7 et Exemple 8).

2.2 Conditionnelles et boucles

Il est possible de spécifier une série de déplacements sous certaines conditions (Exemple 2) . Les deux conditions possibles sont des fonctions permettant de déterminer si une case adjacente donnée est vide ou si elle contient un mur. Elles s'écrivent respectivement `mur <direction>` et `vide <direction>`. Il existe quatre directions possibles : `devant, derriere, à droite, à gauche`.

La négation peut être utilisée devant une condition, avec le mot clé `pas de`. Par exemple, la condition `pas de mur devant` utilisée dans une conditionnelle permet de tester s'il y a un mur devant la tortue.

Une conditionnelle s'écrit de la manière suivante :

```
si <condition>:
    ...
fin si
```

Il est possible d'ajouter un bloc `sinon :` pour spécifier les instructions à effectuer si la condition testée est fausse.

Deux types de boucles sont utilisables. La première permet d'effectuer une série d'instructions tant que la condition testée est vraie et s'utilise de la manière suivante :

```
tant que <condition>:
    ...
fin tant que
```

La seconde permet de répéter n fois une série d'instructions. Comme précédemment, n est un entier positif et peut correspondre à une expression arithmétique. Elle s'utilise de la manière suivante :

```
repete <nombre>:
    ...
fin repete
```

2.3 Fonctions

La définition de fonctions en dehors du programme principal est possible (Exemple 4) . Une fonction se déclare comme suit :

```
fonction <nom>:
    ...
fin fonction
```

Si la fonction prend des arguments en paramètre, l'accès au premier argument dans la fonction se fait avec `$1` et de manière générale, l'argument n se fait avec `$n`. Il n'existe pas de limite au nombre d'arguments d'une fonction.

Il n'est pas possible de créer deux fonctions avec le même nom.

Une fonction est appelée avec son nom et d'éventuels arguments. Par exemple, `test 1 2 3` permet d'appeler une fonction test avec les arguments 1, 2 et 3.

La fonction `main` est le point d'entrée du programme et se déclare comme les autres fonctions.

2.4 Instructions spéciales

Dans le programme principal, il est possible d'utiliser certaines instructions particulières qui modifient les paramètres de l'affichage. Par exemple on peut changer la couleur de la tortue dans le programme principal avec le mot clé `couleur` (Exemple 6). Il existe deux zones de couleurs la `carapace` et la `motif`, si aucune zone n'est précisée la couleur s'applique à la carapace. Les couleurs sont données en hexadécimal au format *#rrggbb*. Par exemple, `couleur carapace #aBcDeF`. On peut spécifier une couleur pour une tortue donnée en ajoutant `@n` à la fin (Exemple 7).

Un nombre[2] de tortues peut être défini par `tortues <nombre>`. Cette instruction est facultative et peut être utilisée au plus une fois. Si elle n'est pas utilisée, il y aura une tortue par défaut.

Le jardin sur lequel déplacer les tortues peut être choisi avec l'instruction `jardin '<fichier>'` (Exemple 7). Ce fichier permet de définir la taille du jardin ainsi que les positions des murs.

Enfin, il est possible d'utiliser des commentaires en les préfixant avec `--`.

2. Peut être une expression arithmétique

3 Interprétation du langage

Afin que vous vous concentriez sur la reconnaissance du langage nous omettons cette section pour l'instant. Une seconde version du sujet avec cette partie vous sera communiquée le jeudi 29 novembre.

4 Exemples

```
fonction main:
    -- ceci est un commentaire
    -- il y a 3 actions de base : avance, recule et saute
    -- ces actions ont un argument facultatif qui est le nombre
    -- de fois où l'action doit être répétée

    avance 1 fois    -- avance 1 fois
    recule           -- recule 1 fois
    saute  2 fois
    recule 2         -- "fois" est facultatif
fin fonction
```

Exemple 1: Exemple de programme avec des actions de déplacement base.

```
fonction main:
    avance 2 fois

    si mur devant: -- une condition
        -- il y a deux fonctions utilisables dans les conditions
        -- mur <direction>      et      vide <direction>
        -- avec <direction> = [à gauche|à droite|devant|derriere]

        tourne à gauche
        -- la tortue tourne d'un quart de cercle à gauche ou à droite

        -- tourne <à gauche|à droite> [nombre de fois]

        avance
        si mur derriere:
            recule
            saute
        sinon: -- il est possible de donner un bloc 'sinon' aux 'si'
            saute
            avance
        fin si
    fin si

    -- comme avance/recule/saute tourne prends un paramètre facultatif
    tourne à gauche 2 fois

    avance 3
fin fonction
```

Exemple 2: Exemple de programme avec des conditions.

```
fonction main:
    repete 5 fois:
        si vide devant:
            avance
        sinon:
            tourne à gauche
        fin si
    fin repete

    recule 2
fin fonction
```

Exemple 3: Exemple de programme avec des répètes.

```
fonction main:
    avance 4 fois

    tant que pas de mur devant:
        -- les conditions peuvent être niées avec "pas de"
        avance
    fin tant que

    -- appel de fonction
    foo 1 2 3
    -- pas de limite au nombre d'argument
    -- /!\ contrairement aux fonctions prédéfinies
    -- /!\ 'foo' n'est pas équivalent à 'foo 1'
fin fonction

fonction foo:       -- il est possible de créer des fonctions
    avance $1 fois
    -- $1, $2, .., $n, font références aux arguments de la fonction
    recule $2
    saute $3 fois
fin fonction
```

Exemple 4: Exemple de programme avec des fonctions.

```
fonction main:
    avance 5*(1+2) fois
    recule 1/(2-3)
    bar 1 2+2 4
fin fonction

-- on peut utiliser des expressions à base de +,-,*,/ et ()
-- dans les arguments des actions/fonctions


fonction bar:
    saute $1 * ($2 + $3)
fin fonction
```

Exemple 5: Exemple de programme avec des expressions arithmétiques.

```
fonction main:
    -- on peut changer la couleur de la tortue
    -- il y a deux couleurs (carapace et motif)

    -- les couleurs sont données en hexadécimal au format #RRGGBB
    -- comme en CSS

    couleur carapace    #aBcDeF
    couleur motif       #282828


    couleur #424242 -- équivalent à 'couleur carapace #424242'


fin fonction
```

Exemple 6: Exemple de programme avec des couleurs.

```
-- on peut gérer plusieurs tortues dans un même script

fonction main:
    tortues 2 -- on définit le nombre de tortues
    jardin '../jardin01.jdn'
    couleur #123456     @1  -- @n  à la fin des instructions permet
    avance 1 fois       @1  -- de choisir la tortue est affectée
    tourne à droite     @2

    avance  -- toutes les tortues sont affectées si rien n'est précisé
fin fonction
```

Exemple 7: Gestion de plusieurs tortues 1.

```
1  -- certaines actions ne peuvent pas avoir d'indications de tortue
2
3  fonction main:
4      tortues 2
5
6      si mur devant @2:
7          -- on peut ajouter l'indication de tortues dans les conditions
8          avance
9          recule  @1
10     fin si
11
12     si pas de vide devant @1:
13         avance
14     fin si
15
16     si mur à gauche:
17         -- si on ne précise pas de tortue, la condition équivaut à
18         -- (mur gauche @1) et (mur gauche @2) et .. et (mur gauche @n)
19         avance
20     fin si
21
22
23 --  tortues 2           @2      -- interdit
24
25 --  si mur devant:      @1      -- interdit
26 --      avance 2
27 --  fin si
28
29 --  repete 4 fois:      @1      -- interdit
30 --      recule
31 --  fin repete
32
33 --  tant que vide derriere:     @3      --interdit
34 --      recule
35 --  fin tant que
36
37 fin fonction
```

Exemple 8: Gestion de plusieurs tortues 2.

```
fonction main:
    avance fois          -- il faut préciser un nombre avant 'fois'

    recule avance            -- une seule instrucion par ligne

    recule 4+                -- expression arithmétique non correcte

    couleur #1010EG      -- pas au format hexadécimal
    couleur 3 fois

    tortues 5 fois       -- pas de 'fois' sur le nombre de tortues

    avance 4 fois       @4 fois -- pas de 'fois' après le @ tortue

    si 4:                    -- 4 n'est pas une condition valide
        avance 3
    fin si

    avance 4 fois + 2 fois   -- ça c'est pas valide

    mur devant               -- une condition n'a rien à faire ici

    avance 0xFF              -- que des nombres en base 10

    si mur devant:           -- pas de 'fin si'
        avance

    fonction foo:    -- une fonction n'a rien à foutre ici
        avance
    fin fonction

fin fonction
```

Exemple 9: Plein de trucs à ne pas faire.