Université de Nice-Sophia Antipolis — Licence d'Informatique — 3e année
Analyses — 2004–2005

TP N° 1

Programmation manuelle d'un analyseur lexical

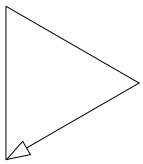
1 Le langage *Logo* 

1.1 Un peu d'historique

Le langage *Logo* a été developpé par Seymour PAPERT en 1967 comme un langage d'apprentissage pour enfants. Le langage permet l'expérimentation de la géométrie planaire par le biais de la programmation, mais a été largement étendu depuis. Dans sa version la plus simple, les déplacements d'une tortue sont commandés par des scripts écrits en *Logo* de manière à dessiner sur l'écran.

Nous allons travailler sur un langage inspiré du langage *Logo*. Par exemple le programme suivant dessine un triangle équilatéral :

```
avancer 100
tourner droite 120d
avancer 100
tourner droite 120d
avancer 100
```



1.2 Les lexèmes du langage

Les lexèmes du langage sont `identificateur`, `mot-clé`, `sens`, `opérateur`, `angle` et `nombre`. L'automate permettant leur reconnaissance est celui de la figure 1 (les mots-clés et sens sont reconnus par le même état de reconnaissance que les identificateurs).

Précisons que les mots-clés sont `tantque`, `si`, `sinon`, `pour`, `poser`, `lever`, `avancer` et `tourner`, tandis que les sens sont `gauche` et `droite`.

2 Question :

Écrivez le programme C de l'analyseur lexical pour ce langage inspiré du langage *Logo*. Il lira sur l'entrée standard un fichier de texte et écrira sur la sortie standard les types des différents lexèmes reconnus. Voici un exemple d'exécution :

Exemple :

– l'entrée standard :

```
lever
avancer 10
titi1 := 7r
titi1:=7r
10toto<<x
tourner # titi1 droite
```

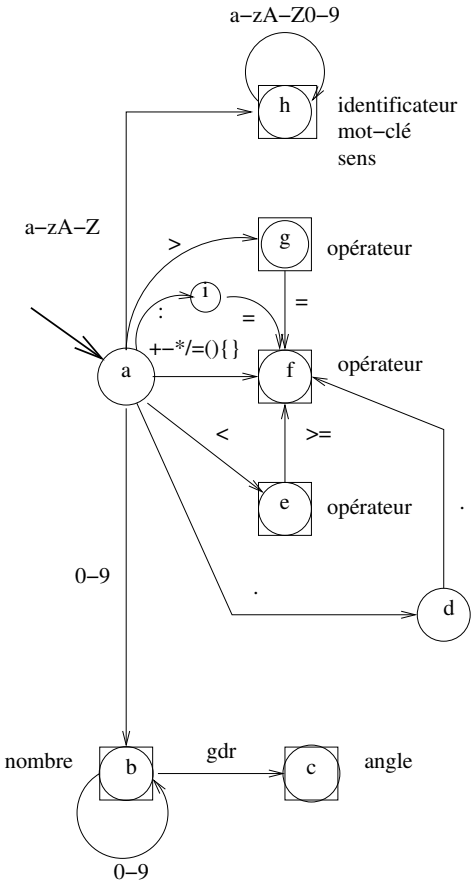


FIG. 1 – L'automate reconnaissant les lexèmes du langage inspiré de *Logo*.

– la sortie standard :

```
mot-clé
mot-clé nombre
ident op angle
ident op angle
nombre ident op op ident
mot-clé ERREUR ident sens
```

```
/* lex_logo.c
   Hand-written lexer for Logo.

   Output:
     succession of ID, KEYWORD, DIRECTION, OP, ANGLE, NUMBER, ERROR and '\n'.

   Input:
     ID        -> LETTER ( LETTER | DIGIT )*
     KEYWORD   -> WHILE | IF | ELSE | FOR | DOWN | UP | FORWARD | TURN
     DIRECTION -> LEFT | RIGHT
     OP        -> SPE | <> | <= | >= | .. | :=
     ANGLE     -> NUMBER [dgr]
     NUMBER    -> DIGIT+
     DIGIT     -> [0-9]
```

```
     LETTER    -> [A-Za-z]
     SPE       -> [+*-/=<>(){}]
*/

#include <stdlib.h>   /* Utility functions, needed for 'exit' */
#include <stdio.h>    /* Input and output, needed for 'fopen' etc. */
#include <ctype.h>    /* Character class tests, needed for 'isalpha' etc. */
#include <string.h>   /* String functions, needed for 'strcasecmp' etc. */
#include <assert.h>   /* Diagnostics, needed for 'assert' */

/* Axioms.  You have to make sure these numbers will not interfere
   with other possible token values, like 'EOF', '\n' etc. */
#define ID        0
#define KEYWORD   1
#define DIRECTION 2
#define OP        3
#define ANGLE     4
#define NUMBER    5
#define ERROR     6

/* Various definitions */
#define OPCHARS   "+-*/=(){}<>.:"
#define ANGLES    "gdr"
#define WHILE     "tantque"
#define IF        "si"
#define ELSE      "sinon"
#define FOR       "pour"
#define DOWN      "poser"
#define UP        "lever"
#define FORWARD   "avancer"
#define TURN      "tourner"
#define LEFT      "gauche"
#define RIGHT     "droite"

/* Define our own 'fgetc' to ease tracing mistakes when 'TRACE' is defined. */
int
fgettok (FILE* f)
{
  int c = fgetc (f);
#ifdef TRACE
  switch (c)
    {
    case EOF:
      printf ("## read: EOF\n"); break;
    case ' ':
      printf ("## read:  \n");   break;
    case '\t':
      printf ("## read: \\t\n"); break;
    case '\n':
      printf ("## read: \\n\n"); break;
    default:
      printf ("## read: %c\n", c);
    }
#endif /* TRACE */
  return c;
}

/* Return the next recognized axiom from the input file 'f'. */
```

```
int
lexer (FILE *f)
{
  int token;
  assert (f != NULL);
  token = fgettok (f);

  /* test for EOF and whitespace */
  if (token == EOF || isspace(token))
    return token;

  /* test for OP */
  if (strchr (OPCHARS, token) != NULL)
    {
      switch (token)
        {
        case ':':
          if (fgettok (f) != '=')
            return ERROR;
          break;
        case '.':
          if (fgettok (f) != '.')
            return ERROR;
          break;
        case '<':
          token = fgettok (f);
          /* Here we are looking ahead to see if we are in an `<>' or
             an `<='.  When looking ahead, one has to remember to
             `ungetc' the lookahead token in case of a failure. */
          if (token != '>' || token != '=')
            ungetc (token, f);
          break;
        case '>':
          token = fgettok (f);
          if (token != '=')
            ungetc (token, f);
        }

      return OP;
    }

  /* test for NUMBER or ANGLE */
  if (isdigit (token))
    {
      /* There is a look ahead operation hidden in this `while' loop. */
      while (isdigit (token = fgettok (f)));

      if (strchr (ANGLES, token))
        return ANGLE;
      else
        {
          ungetc (token, f);
          return NUMBER;
        }
    }

  /* test for ID, KEYWORD and DIRECTION */
  if (isalpha (token))
```

```
    {
      char word[BUFSIZ];
      unsigned int index = 0;

      do
        {
          assert (index < BUFSIZ);
          word[index++] = token;
        }
      while (isalnum (token = fgettok (f)));
      ungetc (token, f);
      assert (index < BUFSIZ);
      word[index] = '\0';

      if (strcasecmp (word, LEFT)            == 0
          || strcasecmp (word, RIGHT)        == 0)
        return DIRECTION;

      else if (strcasecmp (word, WHILE)      == 0
               || strcasecmp (word, IF)      == 0
               || strcasecmp (word, ELSE)    == 0
               || strcasecmp (word, FOR)     == 0
               || strcasecmp (word, DOWN)    == 0
               || strcasecmp (word, UP)      == 0
               || strcasecmp (word, FORWARD) == 0
               || strcasecmp (word, TURN)    == 0)
        return KEYWORD;

      else
        return ID;
    }

  /* otherwise, send an ERROR */
  return ERROR;
}

/* The main returns '0' on a successful lexing, '1' on an erroneous parameter
   number, '2' on an erroneous filename and '3' on an unsuccessful lexing. */
int
main (int argc, char *argv[])
{
  FILE *f;
  int axiom, ret = 0;

  if (argc != 2)
    {
      fprintf (stderr, "Usage: %s <filename>\n", argv[0]);
      exit (1);
    }

  f = fopen (argv[1], "r");
  if (f == NULL)
    {
      perror (argv[1]);
      exit (2);
    }

  while ((axiom = lexer (f)) != EOF)
```

```
    {
      switch (axiom)
        {
        case ID:
          printf ("ident ");   break;
        case KEYWORD:
          printf ("mot-clé "); break;
        case DIRECTION:
          printf ("sens ");    break;
        case OP:
          printf ("op ");      break;
        case ANGLE:
          printf ("angle ");   break;
        case NUMBER:
          printf ("nombre ");  break;
        case ERROR:
          printf ("ERREUR ");
          ret = 3;             break;
        case '\n':
          putchar ('\n');      break;
        }
    }

  fclose (f);
  return ret;
}
```

.....