Un mini Forth en C, Lex, Yacc

27 janvier 2016

cate est un exemple d'utilisation de lex et yacc. C'est un langage à pile, sans variable, avec branchement if-then-else et possibilité de définir des fonctions récursives (fib pour fibonacci, fact pour la factorielle, pgcd ou gcd pour le PGCD). Le seul type disponible est l'entier, et la pile des données ne contient donc que des entiers. *cate* a uniquement des prétentions pédagogiques. Les sources sont dans le répertoire :

`http://ufrsciencestech.u-bourgogne.fr/licence3/LangagesFormels/LEX_YACC/FORTH_en_C/`

Ci-dessous une session, et les sources (moins de 400 lignes) : makefile, cate.h, cate.c, cate.lex, cate.yacc.

1 Une session

```
$ cate
1 2 3 * + .
stack[0]==7

: fact dup if (dup 1 n + fact *) (drop 1) .
6 fact .
stack[1]==720

: carre dup * .
10 carre .
stack[2]==100

: cube dup dup * * .
10 cube .
stack[3]==1000

: pgcd dup  if ( swap 1 cp % pgcd ) (drop) .
36 100 pgcd .
stack[4]==4

: aba swap 1 cp . # a b ... => a b a ...
: gcd dup  if (aba % gcd) (drop) .
36 100 gcd .
stack[6]==4
```

2 Makefile

```
ok: cate forth_en_C.pdf
cate:  cate.c cate.h cate.lex cate.yacc
        lex  cate.lex
        yacc  cate.yacc
```

```
	#g++ -Wno-deprecated -o cate  cate.c -lfl -lc -lm
	g++ -Wno-write-strings -o cate  cate.c -lfl -lc -lm
forth_en_C.pdf : forth_en_C.tex
	latex forth_en_C
	dvipdf forth_en_C
```

3 cate.h

```
#ifndef CAT_INCLUDED
#define CAT_INCLUDED
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>

#define NB 1000
struct Definitions { int nb; char strings[NB][32]; int adresse[NB]; };
extern Definitions definitions;

enum KindDefinition { NIL, SUBR, INT, PAIRE, SI};;
typedef void Function();
struct Subr { Function *fonction; char * nom; } ;
struct Paire { int hd, tl; /* 2 indices dans tabcell */ };
struct Si { int alors, sinon; /* 2 indices dans tabcell */ };
struct Cell { KindDefinition kind;
              union { Subr subr; int entier; Paire paire; Si si_; } cell; } ;

#define NBCELL 100000
extern Cell tabcell[NBCELL];
extern int nbcell;
extern int nil  ;
extern int cons( int tete, int queue);

#define MAXSTACK 10000
extern int  stack[MAXSTACK];
extern int  topstack;

extern int definir_routine( char *nom, Function *f);
extern void empile( int a) ;
extern void routine_plus() ;
extern void routine_mult()  ;
extern void routine_dup();
extern void routine_cp();
extern void routine_swap();
extern void routine_negate() ;
extern void routine_point();
extern void initialise();
extern int alloc_cell(),  cell_entier ( int );
extern int cell_paire( int tete, int queue) ;
extern int cell_si( int alors, int sinon);
extern int cell_by_name( char *str);
extern void evaluer( int cell);
extern void ecrire_cell( int cell);
extern int yyerror( char *s);
extern int yyerror( char *s);
extern int yylex (void);
extern int yyparse (void);
extern int yyerror( char *);
#endif
```

4 cate.c

```
#include"cate.h"
#include"y.tab.c"

Definitions definitions;
Cell tabcell[NBCELL];
int  stack[MAXSTACK];
int  topstack= -1; // la pile est vide
int nbcell= 0;
int nil = 0 ; // -1
int alloc_cell()
{       nbcell++;
        assert( nbcell < NBCELL);
        return nbcell-1;
}
int cell_entier ( int a)
{       int nucell=alloc_cell();
        tabcell[ nucell ].kind = INT;
        tabcell[ nucell ].cell.entier = a;
        return nucell;
}
int cell_paire( int tete, int queue)
{       int nucell=alloc_cell() ;
        tabcell[ nucell ].kind = PAIRE;
        tabcell[ nucell ].cell.paire.hd = tete;
        tabcell[ nucell ].cell.paire.tl = queue;
        return nucell;
}
int cell_si( int alors, int sinon)
{       int cell_si = alloc_cell();
        tabcell[ cell_si].kind = SI;
        tabcell[ cell_si].cell.si_.alors = alors;
        tabcell[ cell_si].cell.si_.sinon = sinon;
        return cell_si;
}
int cell_by_name( char *str)
{       int i, c;
        for(  i=0; i< definitions.nb; i++)
                if (!strcmp( definitions.strings[i], str))
                        return definitions.adresse[i];
        assert(  definitions.nb < NB);
        strcpy( definitions.strings[ definitions.nb] , str);
        c=alloc_cell();
        definitions.adresse[ definitions.nb] = c;
        definitions.nb++;
        return c;
}
int definir_routine( char *nom, Function *f)
{       int nucell= cell_by_name( nom);
        tabcell[ nucell ].kind = SUBR;
        tabcell[ nucell ].cell.subr.fonction= f;
        tabcell[ nucell ].cell.subr.nom= nom;
        return nucell;
}
void empile( int a) {   topstack++; stack[ topstack]=a; }
void routine_plus()
{       assert( topstack >= 1);
        int a=stack[ topstack]; topstack--; int b=stack[ topstack];
        stack[ topstack] = a+b;
}
void routine_mult()
```

```
{       assert( topstack >= 1);
        int a=stack[ topstack]; topstack--; int b=stack[ topstack];
        stack[ topstack] = a*b;
}
void routine_gt()
{       assert( topstack >= 1);
        int b=stack[ topstack]; topstack--; int a=stack[ topstack];
        stack[ topstack] = (a > b);
}
void routine_geq()
{       assert( topstack >= 1);
        int b=stack[ topstack]; topstack--; int a=stack[ topstack];
        stack[ topstack] = (a >= b);
}
void routine_lt()
{       assert( topstack >= 1);
        int b=stack[ topstack]; topstack--; int a=stack[ topstack];
        stack[ topstack] = (a < b);
}
void routine_leq()
{       assert( topstack >= 1);
        int b=stack[ topstack]; topstack--; int a=stack[ topstack];
        stack[ topstack] = (a <= b);
}
void routine_eq()
{       assert( topstack >= 1);
        int b=stack[ topstack]; topstack--; int a=stack[ topstack];
        stack[ topstack] = (a == b);
}
void routine_exp()
{       int r= 1; int i;
        assert( topstack >= 1);
        int b=stack[ topstack]; topstack--; int a=stack[ topstack];
        for( i=0; i< b; i++) r *= a;
        stack[ topstack] = r;
}
void routine_quotient()
{       int r= 1; int i;
        assert( topstack >= 1);
        int b=stack[ topstack]; topstack--; int a=stack[ topstack];
        stack[ topstack] = a / b;
}
void routine_modulo()
{       int r= 1; int i;
        assert( topstack >= 1);
        int b=stack[ topstack]; topstack--; int a=stack[ topstack];
        stack[ topstack] = a % b;
}
void routine_dup()
{       assert( topstack >= 0);
        int a=stack[ topstack]; topstack++; stack[ topstack] = a;
}
void routine_cp()
{       int k=stack[ topstack];
        assert( topstack -k-1 >= 0 );
        stack[ topstack] = stack[ topstack-k-1];
}
void routine_drop()
{       assert( topstack >= 1);
        topstack-- ;
}
void routine_swap()
```

```
{       assert( topstack >= 1);
        int a=stack[ topstack];
        int b=stack[ topstack-1];
        stack[ topstack] = b;
        stack[ topstack-1] = a;
}
void routine_negate()
{       assert( topstack >= 0);
        int a=stack[ topstack];
        stack[ topstack] = 0 - a;
}
void routine_reset() { topstack = -1 ; }
void routine_point()
{       assert( topstack >= 0);
        printf( "= %d\n", stack[ topstack]);
}
void ecrire_cell_bis( int cell, int level)
// ca evite de planter pour les programmes recursifs
{       if (!level) { printf( "..."); return; }
        int p= cell;
        switch (tabcell[p].kind)
        { case SUBR: printf("%s ", tabcell[p].cell.subr.nom); break;
          case INT : printf("%d ", tabcell[p].cell.entier); break;
          case PAIRE: printf("( ");
        {     int q;
              ecrire_cell_bis( tabcell[p].cell.paire.hd, level-1);
              for( q=tabcell[p].cell.paire.tl;
                   q != nil && tabcell[q].kind == PAIRE;
                   q=tabcell[q].cell.paire.tl)
                 ecrire_cell_bis( tabcell[q].cell.paire.hd, level-1);
              if (q==nil)  printf(") ");
              else { printf(". ");
                     ecrire_cell_bis( tabcell[p].cell.paire.tl, level-1);
                     printf(") ");
                   }
              break;
        }
        case SI : printf("if ");
                    ecrire_cell_bis( tabcell[p].cell.si_.alors, level-1);
                    ecrire_cell_bis( tabcell[p].cell.si_.sinon, level-1);
                   // printf("FINSI ");
                    break;
        case NIL :  printf("nil ");  break;
        default: break;
} }
void ecrire_cell( int cell) { ecrire_cell_bis( cell, 4) ; printf("\n"); }
void evaluer( int prog)
{       if( prog==nil) return;
        switch (tabcell[prog].kind)
        { case SUBR: (*tabcell[prog].cell.subr.fonction)(); break;
          case INT : empile( tabcell[prog].cell.entier); break;
          case PAIRE: evaluer( tabcell[prog].cell.paire.hd);
                      evaluer( tabcell[prog].cell.paire.tl); break;
          case SI : if (stack[ topstack] )
                { topstack--;  evaluer( tabcell[prog].cell.si_.alors); }
                else
                { topstack--; evaluer( tabcell[prog].cell.si_.sinon); }
                break;
         case NIL : break;
         default: break;
        }
}
```

```
void initialise()
{       assert( 0==cell_by_name( "nil"));
        tabcell[0].kind = NIL;
        definir_routine( "+", routine_plus);
        definir_routine( "/", routine_quotient);
        definir_routine( "%", routine_modulo);
        definir_routine( "*", routine_mult);
        definir_routine( "n", routine_negate);
        definir_routine( ".", routine_point);
        definir_routine( "^", routine_exp);
        definir_routine( "cp", routine_cp);
        definir_routine( "dup", routine_dup);
        definir_routine( "swap", routine_swap);
        definir_routine( "drop", routine_drop);
        definir_routine( "<=", routine_leq);
        definir_routine( ">=", routine_geq);
        definir_routine( ">", routine_gt);
        definir_routine( "<", routine_lt);
        definir_routine( "=", routine_eq);
        definir_routine( "reset", routine_reset);
}
int main( int argc, char **argv) { initialise(); yyparse(); }
```

5 cate.lex

```
%{
#include "cate.h"
%}
chiffre [0-9]
lettre [a-zA-Z]
%%
"␣"|\t|\n      {/* sauter les espaces */}
"#".*          { /* commentaire */ }
":"            { return DEF; }
"\."           { return PT; }
";"            { return PV; }
"("            { return LP; }
")"            { return RP; }
"if"           { return IF; }
{chiffre}+     { yylval=atoi(yytext); return INTEGER; }
"\+"           { yylval= cell_by_name( "+");  return WORD; }
"\*"           { yylval= cell_by_name( "*");  return WORD; }
"\^"           { yylval= cell_by_name( "^");  return WORD; }
"%"            { yylval= cell_by_name( "%");  return WORD; }
"\/"           { yylval= cell_by_name( "/");  return WORD; }
"<="           { yylval= cell_by_name( "<=");  return WORD; }
">="           { yylval= cell_by_name( ">=");  return WORD; }
"<"            { yylval= cell_by_name( "<");  return WORD; }
">"            { yylval= cell_by_name( ">");  return WORD; }
"="            { yylval= cell_by_name( "=");  return WORD; }
{lettre}({lettre}|{chiffre})* { yylval= cell_by_name( yytext); return WORD; }
.              { fprintf(stderr,"LEX:␣caractere␣inattendu:%c\n", *yytext);
}
%%
```

6 cate.yacc

```
%{
#include"cate.h"
%}
```

```
%token DEF IF
%token INTEGER
%token WORD
%token LP RP
%token  PT PV
%start prog
%%
prog : {} ;
prog : commandes PV { ecrire_cell( $1);
            evaluer( $1);
            if (topstack>=0)
                printf( "stack[%d]==%d\n", topstack, stack[topstack]);
            else printf( "error, empty stack\n"); }  prog  ;
prog : commandes PT { /* ecrire_cell( $1);  */
            evaluer( $1);
            if (topstack>=0)
                printf( "stack[%d]==%d\n", topstack, stack[topstack]);
            else printf( "error, empty stack\n"); }  prog  ;
prog : definition PT {} prog  ;
prog : definition PV { printf("definition:\n");
                       ecrire_cell( $1); $$=$1; } prog  ;
commandes : { $$=nil; } | commande commandes { $$= cell_paire( $1, $2); } ;
commande : INTEGER { $$= cell_entier( $1); } ;
commande : WORD { $$= $1; };
commande : IF LP commandes RP LP commandes RP { $$= cell_si( $3, $6); } ;
definition : DEF WORD commandes
        { tabcell[ $2].kind=PAIRE;
          tabcell[ $2].cell.paire.hd=tabcell[$3].cell.paire.hd;
          tabcell[ $2].cell.paire.tl=tabcell[$3].cell.paire.tl;
          $$ =  $2;
        };
%%
#include "lex.yy.c"
int yyerror( char *s)
{
    printf( "%s\n", s);
    return 0;
}
```