DEMO : un langage d'exemple

H. Toussaint
hto@info.fundp.ac.be

Le 5 octobre 2011

Table des matières

1 Introduction

DEMO est un langage de calcul simple construit dans le seul but de vous fournir un exemple complet de 'compilateur'. Il se résume à imiter le comportement d'une calculatrice entière à mémoires multiples.

2 Syntaxe

Un programme DEMO se résume à une suite d'instructions simples :

```
DEMO := Instruction*
```

Les seules instructions acceptées sont : l'affichage d'une valeur (print), la lecture d'une entrée de l'utilisateur (read) et l'affectation (affect).

```
Instruction := print(Expr) | read(Expr) | affect(VAR, Expr)
```

Les expressions calculables se résument aux additions, soustractions, multiplications et division modulo.

```
Expr := NB | VAR | Expr '+' Expr | Expr '-' Expr | Expr '*' Expr | Expr 'mod' Expr

VAR := [a-zA-Z]+
NB := 0 | [1-9][0-9]*
```

3 Sémantique

Les règles habituelles de sémantique sont d'usage, à une exception près : en DEMO, les variables ne doivent pas être déclarées avant d'être utilisées ; elles le sont automatiquement au moment de leur usage.

Les règles de priorité entre les opérations sont les règles usuelles (">" signifie "est plus prioritaire que") :

$$\boxed{\text{mod} > * > -,+}$$

4 Exemple

Soit le programme suivant :

```
affect(x,9)
affect(y,5)
affect(x, 9-y+6*3 mod (4+x))
read(y)
print(x+y)
```

Le pcode produit sera (toutes les lignes commencant par un point-virgule sont des commentaires) :

```
; *** DEMO compiler
; *** H. Toussaint (hto@info.fundp.ac.be), 14/06/05
;
;*** BEGIN yyparse() ***
;*** END yyparse() ***
;*** BEGIN printTree(..) ***
; [0x8052390] type=AT_ROOT, ival=0, sval=NULL, left=0x8052378, right=(nil)
; [0x8052378] type=AT_ILIST, ival=0, sval=NULL, left=0x80522e0, right=0x8052360
; [0x80522e0] type=AT_ILIST, ival=0, sval=NULL, left=0x8052288, right=0x80522c8
; [0x8052288] type=AT_ILIST, ival=0, sval=NULL, left=0x8052108, right=0x8052270
; [0x8052108] type=AT_ILIST, ival=0, sval=NULL, left=0x8052098, right=0x80520f0
; [0x8052098] type=AT_ILIST, ival=0, sval=NULL, left=(nil), right=0x8052080
; [0x8052080] type=AT_OPAFF, ival=0, sval=NULL, left=0x8052050, right=0x8052068
; [0x8052050] type=AT_VAR, ival=0, sval=’x’, left=(nil), right=(nil)
; [0x8052068] type=AT_NB, ival=9, sval=NULL, left=(nil), right=(nil)
; [0x80520f0] type=AT_OPAFF, ival=0, sval=NULL, left=0x80520c0, right=0x80520d8
; [0x80520c0] type=AT_VAR, ival=0, sval=’y’, left=(nil), right=(nil)
; [0x80520d8] type=AT_NB, ival=5, sval=NULL, left=(nil), right=(nil)
; [0x8052270] type=AT_OPAFF, ival=0, sval=NULL, left=0x8052130, right=0x8052258
; [0x8052130] type=AT_VAR, ival=0, sval=’x’, left=(nil), right=(nil)
; [0x8052258] type=AT_OPMOD, ival=0, sval=NULL, left=0x80521e8, right=0x8052240
; [0x80521e8] type=AT_OPADD, ival=0, sval=NULL, left=0x8052188, right=0x80521d0
; [0x8052188] type=AT_OPSUB, ival=0, sval=NULL, left=0x8052148, right=0x8052170
; [0x8052148] type=AT_NB, ival=9, sval=NULL, left=(nil), right=(nil)
; [0x8052170] type=AT_VAR, ival=0, sval=’y’, left=(nil), right=(nil)
; [0x80521d0] type=AT_OPMUL, ival=0, sval=NULL, left=0x80521a0, right=0x80521b8
; [0x80521a0] type=AT_NB, ival=6, sval=NULL, left=(nil), right=(nil)
; [0x80521b8] type=AT_NB, ival=3, sval=NULL, left=(nil), right=(nil)
; [0x8052240] type=AT_OPADD, ival=0, sval=NULL, left=0x8052200, right=0x8052228
; [0x8052200] type=AT_NB, ival=4, sval=NULL, left=(nil), right=(nil)
; [0x8052228] type=AT_VAR, ival=0, sval=’x’, left=(nil), right=(nil)
; [0x80522c8] type=AT_OPREAD, ival=0, sval=NULL, left=0x80522b0, right=(nil)
; [0x80522b0] type=AT_VAR, ival=0, sval=’y’, left=(nil), right=(nil)
; [0x8052360] type=AT_OPPRINT, ival=0, sval=NULL, left=0x8052348, right=(nil)
; [0x8052348] type=AT_OPADD, ival=0, sval=NULL, left=0x8052308, right=0x8052330
; [0x8052308] type=AT_VAR, ival=0, sval=’x’, left=(nil), right=(nil)
; [0x8052330] type=AT_VAR, ival=0, sval=’y’, left=(nil), right=(nil)
;*** END printTree(..) ***
;*** BEGIN SymbolTable ***
;*** END SymbolTable ***
;*** BEGIN printSymbolTable(..) ***
; [0x80523a8] id=’x’, location=-1, next=0x80523b8
; [0x80523b8] id=’y’, location=-1, next=0x80523c8
;*** END printSymbolTable(..) ***
;*** BEGIN computeLocations(..) ***
;*** END computeLocations(..) ***
```

```
;*** BEGIN printSymbolTable(..) +locations ***
; [0x80523a8] id=’x’, location=2, next=0x80523b8
; [0x80523b8] id=’y’, location=3, next=0x80523c8
;*** END printSymbolTable(..) +locations ***
;*** BEGIN PCodeGeneration ***
; ssp 2 + memory used for variables (2 temporary locations for modulo algorithm)
ssp 4

; begin of affectation 7
; loading ADRESS of variable ’x’
lda i 0 2
; loading constant value ’9’
ldc i 9
sto i
; end of affectation 7

; begin of affectation 9
; loading ADRESS of variable ’y’
lda i 0 3
; loading constant value ’5’
ldc i 5
sto i
; end of affectation 9

; begin of affectation 11
; loading ADRESS of variable ’x’
lda i 0 2

; begin of modulo 12

; saves computed arguments
; saving x value
lda i 0 0

; begin of sum 13

; begin of substraction 14
; loading constant value ’9’
ldc i 9
; loading VALUE of variable ’y’
lda i 0 3
ind i
sub i
; end of substraction 14

; begin of multiplication 17
; loading constant value ’6’
ldc i 6
; loading constant value ’3’
ldc i 3
mul i
; end of the multiplication 17
add i
; end of sum 13
sto i
; done with saving x value

; saving y value
```

```
lda i 0 1

; begin of sum 20
; loading constant value ’4’
ldc i 4
; loading VALUE of variable ’x’
lda i 0 2
ind i
add i
; end of sum 20
sto i
; done with saving y value


; checks if y == 0
lda i 0 1
ind i
ldc i 0
equ i
fjp @mod_not_zero_12
; mod division by 0 !!
stp

; begin of mod inner loop
define @mod_not_zero_12

; !(x < y) ?
lda i 0 0
ind i
lda i 0 1
ind i
les i
not b
fjp @mod_end_12

; loop body : x := x-y
lda i 0 0
lda i 0 0
ind i
lda i 0 1
ind i
sub i
sto i
ujp @mod_not_zero_12

; end, put result on top of the stack
define @mod_end_12
lda i 0 0
ind i

; end of modulo 12
sto i
; end of affectation 11

; begin of read 23
; loading ADRESS of variable ’y’
lda i 0 3
read
```

```
sto i
; end of read 23

; begin of print 24

; begin of sum 25
; loading VALUE of variable 'x'
lda i 0 2
ind i
; loading VALUE of variable 'y'
lda i 0 3
ind i
add i
; end of sum 25
prin
; end of print 24
; end of program
stp
;*** END PCodeGeneration ***
;*** BEGIN Cleaning ***
;*** END Cleaning ***
```

5 Sources

Les sources données ici ont pour unique but de vous permettre de mieux comprendre les mécanismes de communication entre les divers éléments composant un compilateur : lex, yacc, arbre syntaxique, table des symboles et fonctions génératrices de pcode.

5.1 demo.l

```
/* demo.l
 *
 * part of the DEMO compiler
 *
 * H. Toussaint (hto@info.fundp.ac.be), 14/06/05
 */

%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ast.h"
#include "y.tab.h"

int num_lines=1;
%}

%option noyywrap

var       [A-Za-z]*
nbr       (0|[1-9][0-9]*)
blank     [ \t]+

%%

"print"      {return PRINT;}
```

```
"read"       {return READ;}
"affect"     {return AFFECT;}

"+"          {return PLUS;}
"-"          {return MINUS;}
"*"          {return TIMES;}
"("          {return LP;}
")"          {return RP;}
","          {return COMMA;}
"mod"        {return MODULO;}

{var}        {
                yylval.sval=(char*)calloc(strlen(yytext)+1,sizeof(char));
                strcpy(yylval.sval,yytext);
                return VAR;
             }

{nbr}        {
                yylval.ival=atoi(yytext);
                return NB;
             }

"\n"         {++num_lines;}

{blank}      {/*On passe*/}

.            {
                fprintf(stderr,"KO\n");
                printf("ERROR : invalid '%s' in line %d\n",yytext, num_lines);
                exit(0);
             }

%%
```

5.2 demo.y

```
/* demo.y
 *
 * part of the DEMO compiler
 *
 * H. Toussaint (hto@info.fundp.ac.be), 14/06/05
 */

%{
#include <stdio.h>
#include <stdlib.h>

#include "ast.h"
#include "sym.h"
#include "pcode.h"

extern int num_lines;
extern char* yytext;


// to avoid 'implicit definition'
int yylex(void);
int yyerror(char *str);
ASTTREE root;
%}


// définition du type des variables $x
%union{
        int ival;
        char *sval;
        ASTTREE tval;
}

%token READ PRINT AFFECT
%token LP RP COMMA PLUS MINUS TIMES
%token VAR NB

%left MODULO
%left PLUS MINUS
%left TIMES


%type <tval> DEMO InstructionList Instruction Expr Var

%%

DEMO : InstructionList { root = createNode(AT_ROOT, 0, NULL, $1, NULL); }
;

InstructionList :                             { $$ = NULL; }
                | InstructionList Instruction { $$ = createNode(AT_ILIST, 0, NULL, $1, $2);}
;

Instruction : PRINT LP Expr RP            {$$ = createNode(AT_OPPRINT, 0, NULL, $3, NULL);}
            | READ  LP Var RP             {$$ = createNode(AT_OPREAD, 0, NULL, $3, NULL);}
            | AFFECT LP Var COMMA Expr RP {$$ = createNode(AT_OPAFF, 0, NULL, $3, $5);}
;

Expr : NB               {$$ = createNode(AT_NB, yylval.ival, NULL, NULL, NULL);}
     | Var              {$$ = $1;}
     | Expr PLUS Expr   {$$ = createNode(AT_OPADD, 0, NULL, $1, $3);}
     | Expr MINUS Expr  {$$ = createNode(AT_OPSUB, 0, NULL, $1, $3);}
     | Expr TIMES Expr  {$$ = createNode(AT_OPMUL, 0, NULL, $1, $3);}
     | LP Expr RP       {$$ = $2;}
     | Expr MODULO Expr {$$ = createNode(AT_OPMOD, 0, NULL, $1, $3);}
;

Var : VAR { $$ = createNode(AT_VAR, 0, yylval.sval, NULL, NULL);}
;

%%

int yyerror(char *str)
{
        fprintf(stderr,"KO\n");
        printf("ERROR '%s' AT LINE  %d : UNRECOGNISED '%s'\n",
                str,num_lines,yytext);
        exit(0);
}
```

```
int main()
{
  SYMTABLE sym;

  printf("; *** DEMO compiler\n");
  printf("; *** H. Toussaint (hto@info.fundp.ac.be), 14/06/05\n");
  printf(";\n");

  printf(";*** BEGIN yyparse() ***\n");
  yyparse();
  printf(";*** END yyparse() ***\n");

  printf(";*** BEGIN printTree(..) ***\n");
  printTree(root);
  printf(";*** END printTree(..) ***\n");

  printf(";*** BEGIN SymbolTable ***\n");
  sym = createSymbolTable();
  fillSymbolTable(root, sym);
  printf(";*** END SymbolTable ***\n");

  printf(";*** BEGIN printSymbolTable(..) ***\n");
  printSymbolTable(sym);
  printf(";*** END printSymbolTable(..) ***\n");

  printf(";*** BEGIN computeLocations(..) ***\n");
  computeLocations(sym);
  printf(";*** END computeLocations(..) ***\n");

  printf(";*** BEGIN printSymbolTable(..) +locations ***\n");
  printSymbolTable(sym);
  printf(";*** END printSymbolTable(..) +locations ***\n");

  printf(";*** BEGIN PCodeGeneration ***\n");
  pcodeGenValue(root, sym);
  printf(";*** END PCodeGeneration ***\n");

  printf(";*** BEGIN Cleaning ***\n");
  freeNode(root);
  freeSymbolTable(sym);
  printf(";*** END Cleaning ***\n");

  fprintf(stderr,"OK\n");

  return 0;
}
```

5.3 ast.h

```
/* ast.h
 *
 * part of the DEMO compiler
 *
 * H. Toussaint (hto@info.fundp.ac.be), 14/06/05
 */

#ifndef AST_H
#define AST_H
```

```
// do not change values below, or also edit humanReadableNodeType() in ast.c
#define AT_VAR     0
#define AT_NB      1
#define AT_OPADD   2
#define AT_OPSUB   3
#define AT_OPMUL   4
#define AT_OPPRINT 5
#define AT_OPREAD  6
#define AT_OPAFF   7
#define AT_OPMOD   8
#define AT_ILIST   9
#define AT_ROOT   10


struct astnode {
  int type;

  int ival;
  char* sval;

  struct astnode * left;
  struct astnode * right;
};

typedef struct astnode * ASTTREE;
typedef struct astnode   ASTNODE;

extern ASTTREE createNode(int type, int ival, char* sval, ASTTREE left, ASTTREE right);
extern void freeNode(ASTTREE node);
extern void freeTree(ASTTREE tree);

extern void printTree(ASTTREE tree);

#endif
```

5.4 sym.h

```
/* sym.h
 *
 * part of the DEMO compiler
 *
 * H. Toussaint (hto@info.fundp.ac.be), 14/06/05
 */

#ifndef SYM_H
#define SYM_H

struct _stitem {
  char* id;
  int location;

  struct _stitem* next;
};

typedef struct _stitem STITEM;
typedef STITEM * SYMTABLE;


extern SYMTABLE createSymbolTable();
extern void freeSymbolTable(SYMTABLE s);

extern int addToSymbolTable(SYMTABLE s, char* name);
extern int alreadyIsSymbol(SYMTABLE s, char* name);

extern int computeLocations(SYMTABLE s);
extern int getLocation(SYMTABLE s, char* name);
extern int getMaxMemoryUsage(SYMTABLE s);

extern void printSymbolTable(SYMTABLE s);
#endif
```

5.5 pcode.h

```
/* pcode.h
 *
 * part of the DEMO compiler
 *
 * H. Toussaint (hto@info.fundp.ac.be), 14/06/05
 */

#ifndef _PCODE_H
#define _PCODE_H

#include "ast.h"
#include "sym.h"

extern int fillSymbolTable(ASTTREE tree, SYMTABLE s);
extern int pcodeGenAddress(ASTTREE tree, SYMTABLE s);
extern int pcodeGenValue(ASTTREE tree, SYMTABLE s);
#endif
```

5.6 ast.c

```
/* ast.c
 *
 * part of the DEMO compiler
 *
 * H. Toussaint (hto@info.fundp.ac.be), 14/06/05
 */

#include <stdio.h>
#include <stdlib.h>

#include "ast.h"

ASTTREE createNode(int type,
                   int ival,
                   char* sval,
                   ASTTREE left,
                   ASTTREE right)
{
  ASTTREE node = (ASTTREE) malloc(sizeof(ASTNODE));
  if (node == NULL)
```

```
    {
      fprintf(stderr,"KO\n");
      printf("ERROR : malloc failed in createNode(..)\n");
      exit(1);
    }
  else
    {
      node->type = type;
      node->ival = ival;
      node->sval = sval;
      node->left = left;
      node->right = right;

      return node;
    }
}

void freeNode(ASTTREE node)
{
  if (node != NULL)
    {
      if (node->sval != NULL) free(node->sval);
      if (node->left != NULL) freeNode(node->left);
      if (node->right != NULL) freeNode(node->right);

      free(node);
    }
}

void freeTree(ASTTREE tree) // idem above but top root is static
{
  if (tree != NULL)
    {
      if (tree->sval != NULL) free(tree->sval);
      if (tree->left != NULL) freeNode(tree->left);
      if (tree->right != NULL) freeNode(tree->right);
    }
}


char* humanReadableNodeType(int type)
{
  switch(type) {

  case AT_VAR:      return "AT_VAR"; break;
  case AT_NB:       return "AT_NB"; break;
  case AT_OPADD:    return "AT_OPADD"; break;
  case AT_OPSUB:    return "AT_OPSUB"; break;
  case AT_OPMUL :   return "AT_OPMUL"; break;
  case AT_OPPRINT : return "AT_OPPRINT"; break;
  case AT_OPREAD :  return "AT_OPREAD"; break;
  case AT_OPAFF :   return "AT_OPAFF"; break;
  case AT_ILIST :   return "AT_ILIST"; break;
  case AT_OPMOD:    return "AT_OPMOD"; break;
  case AT_ROOT:     return "AT_ROOT"; break;
  default :         return "??";
  }
}
```

```
void printTree(ASTTREE tree)
{
  if (tree != NULL)
    {
      printf("; [%p] type=%s, ival=%d, sval=", tree, humanReadableNodeType(tree->type), tree->ival);
      if (tree->sval == NULL) printf("NULL");
      else printf("’%s’", tree->sval);
      printf(", left=%p, right=%p\n", tree->left, tree->right);

      printTree(tree->left);
      printTree(tree->right);
    }
}
```

5.7 sym.c

```
/* sym.c
 *
 * part of the DEMO compiler
 *
 * H. Toussaint (hto@info.fundp.ac.be), 14/06/05
 */

#include <stdio.h>
#include <strings.h>
#include <stdlib.h>

#include "sym.h"

#define NO_LOC -1

STITEM * createSTNode()
{
  STITEM * node = (STITEM * ) malloc(sizeof(STITEM));

  if (node == NULL)
    {
      fprintf(stderr,"KO\n");
      printf("ERROR : cannot malloc in createSTNode()\n");
      exit(1);
    }

  node->id = NULL;
  node->location = NO_LOC;
  node->next = NULL;

  return node;
}

SYMTABLE createSymbolTable()
{
  SYMTABLE s = (SYMTABLE) createSTNode();
  return s;
}

void freeSymbolTable(SYMTABLE s)
{
  if (s != NULL)
    {
      //if (s->id != NULL) free(s->id);
      freeSymbolTable(s->next);
      free(s);
    }
}

STITEM * symbolLookup(SYMTABLE s, char* name)
{
  if (s == NULL) return NULL;
  else
    if (s->next == NULL) return NULL;
    else
      {
        if (strcmp(s->id, name) == 0) return s;
        else return symbolLookup(s->next, name);
      }
}

int alreadyIsSymbol(SYMTABLE s, char* name)
{
  return (symbolLookup(s,name) == NULL) ? 0 : 1;
}

int addToSymbolTable(SYMTABLE s, char* name)
{
  if (alreadyIsSymbol(s,name)) return 0;
  else
    {
      while (s->next != NULL) s = s->next;

      s->id = name;
      s->next = createSTNode();

      return 1;
    }
}

int computeLocations(SYMTABLE s)
{
  SYMTABLE local = s;
  int available = 2; /* first available mem cell is 2,
                      * because 0 and 1 are used by modulo algorithm */

  while (local != NULL) {
    if (local->next != NULL)
      {
        local->location = available;

        available++;
      }
    local = local->next;
  }
}

int getLocation(SYMTABLE s, char* name)
{
  STITEM * node = symbolLookup(s, name);

  if (node == NULL) return -1;
  else
    {
      if (node->location == NO_LOC)
        // need to compute locations before using them
        computeLocations(s);

      return node->location;
    }
}

int getMaxMemoryUsage(SYMTABLE s)
{
  SYMTABLE tmp = s;
  int max = 0;

  while (tmp != NULL)
    {
      max = (max < tmp->location+1) ? tmp->location+1 : max;
      tmp = tmp->next;
    }
  return max;
}

void printSymbolTable(SYMTABLE s)
{
  if (s != NULL)
    {
      if (s->next != NULL)
        {
          printf("; [%p] id=", s);
          if (s->id == NULL) printf("NULL");
          else printf("’%s’", s->id);
          printf(", location=%d, next=%p\n", s->location, s->next);

          printSymbolTable(s->next);
        }
    }
}
```

5.8 pcode.c

```
/* pcode.c
 *
 * part of the DEMO compiler
 *
 * H. Toussaint (hto@info.fundp.ac.be), 14/06/05
 */

#include <stdio.h>

#include <stdlib.h>

#include "pcode.h"

int fillSymbolTable(ASTTREE tree, SYMTABLE s)
```

```
{
  if (tree == NULL)
    return 0;

  if (tree->type == AT_VAR)
    {
      if (!alreadyIsSymbol(s, tree->sval))
        addToSymbolTable(s, tree->sval);
    }

  fillSymbolTable(tree->left, s);
  fillSymbolTable(tree->right, s);

  return 0;
}

int pcodeGenAddress(ASTTREE tree, SYMTABLE s)
{
  STITEM* node;
  int location;

  if (tree == NULL)
    return 0;

  switch (tree->type) {

  case AT_VAR: // variable
    location = getLocation(s, tree->sval);
    if (location < 0)
      {
        // this should NOT happen, since it will cause havoc on adress space
        fprintf(stderr,"KO\n");
        printf("ERROR : (!!) pcodeGenAddress : VAR ’%s’ has no location\n", tree->sval);
        exit(1);
      }
    else
      printf("; loading ADRESS of variable ’%s’\n", tree->sval);
      printf("lda i 0 %d\n",location);
    break;

  default:
    fprintf(stderr,"KO\n");
    printf("ERROR : unrecognized type=%d in pcodeGenAddress(..)\n", tree->type);
    exit(1);
  }

  return 0;
}


int pcodeGenValue(ASTTREE tree, SYMTABLE s)
{
  STITEM* node;
  int location;
  static int staticlabel = 0;
  int label = staticlabel;

  staticlabel++;
  if (tree == NULL)
    return 0;

  switch (tree->type) {

  case AT_ROOT:
    printf("; ssp 2 (0 & 1) + memory used for variables (2 temporary locations for modulo algorithm)\n");
    printf("ssp %d\n", (getMaxMemoryUsage(s) <= 0) ? 2 : getMaxMemoryUsage(s) );
    pcodeGenValue(tree->left,s);
    printf("; end of program\n");
    printf("stp\n");
    break;

  case AT_VAR: // variable
    location = getLocation(s, tree->sval);
    if (location < 0)
      {
        // this should NOT happen, since it will cause havoc on address space
        fprintf(stderr,"KO\n");
        printf("ERROR : (!!) pcodeGenValue : VAR '%s' has no location\n", tree->sval);
        exit(1);
      }
    else
      printf("; loading VALUE of variable '%s'\n", tree->sval);
      printf("lda i 0 %d\nind i\n",location);
    break;

  case AT_NB: // raw number
    printf("; loading constant value '%d'\n", tree->ival);
    printf("ldc i %d\n", tree->ival);
    break;

  case AT_OPADD: // sum
    printf("\n; begin of sum %d\n", label);
    pcodeGenValue(tree->left, s);
    pcodeGenValue(tree->right, s);
    printf("add i\n");
    printf("; end of sum %d\n", label);
    break;

  case AT_OPSUB: // substraction
    printf("\n; begin of substraction %d\n", label);
    pcodeGenValue(tree->left, s);
    pcodeGenValue(tree->right, s);
    printf("sub i\n");
    printf("; end of substraction %d\n", label);
    break;

  case AT_OPMUL: // multiplication
    printf("\n; begin of multiplication %d\n", label);
    pcodeGenValue(tree->left, s);
    pcodeGenValue(tree->right, s);
    printf("mul i\n");
    printf("; end of the multiplication %d\n", label);
    break;

  case AT_OPPRINT: // print command
```

```
      printf("\n; begin of print %d\n", label);
      pcodeGenValue(tree->left, s);
      printf("prin\n");
      printf("; end of print %d\n", label);
      break;

    case AT_OPREAD: // read command
      printf("\n; begin of read %d\n", label);
      pcodeGenAddress(tree->left, s);
      printf("read\nsto i\n");
      printf("; end of read %d\n", label);
      break;

    case AT_OPAFF:
      printf("\n; begin of affectation %d\n", label);
      pcodeGenAddress(tree->left, s);
      pcodeGenValue(tree->right, s);
      printf("sto i\n");
      printf("; end of affectation %d\n", label);
      break;

    case AT_OPMOD:

      /* x mod y  (x,y >= 0)
       *   put y into cell 1 so we can easily remember it
       *   put x into cell 0 so we can easily update it
       *
       * x mod y = 'ERROR' if (y==0)
       *           'x after while !(x < y) x := x-y' else
       */

      printf("\n; begin of modulo %d\n", label);

      printf("\n; saves computed arguments\n");

      /* x */
      printf("; saving x value\n");
      printf("lda i 0 0\n");
      pcodeGenValue(tree->left, s);
      printf("sto i\n");
      printf("; done with saving x value\n\n");

      /* y */
      printf("; saving y value\n");
      printf("lda i 0 1\n");
      pcodeGenValue(tree->right, s);
      printf("sto i\n");
      printf("; done with saving y value\n\n");

      /* checks if y == 0 */
      printf("\n; checks if y == 0\n");
      printf("lda i 0 1\n");
      printf("ind i\n");
      printf("ldc i 0\n");
      printf("equ i\n");
      printf("fjp @mod_not_zero_%d\n", label);

      /* here y == 0 */
      printf("; mod division by 0 !!\n");
      printf("stp\n");

      /* else y != 0 */
      printf("\n; begin of mod inner loop\n");
      printf("define @mod_not_zero_%d\n",label);

      /* checks if x < y  */
      printf("\n; !(x < y) ?\n");
      printf("lda i 0 0\nind i\n");
      printf("lda i 0 1\nind i\n");
      printf("les i\n");
      printf("not b\n");
      printf("fjp @mod_end_%d\n",label);

      /* !(x<y) -> x := x-y ; goto @mod_not_zero_%d */
      printf("\n; loop body : x := x-y\n");
      printf("lda i 0 0\n");
      printf("lda i 0 0\nind i\n");
      printf("lda i 0 1\nind i\n");
      printf("sub i\n");
      printf("sto i\n");

      printf("ujp @mod_not_zero_%d\n", label);

      /* end -> put mod result on top of the stack */
      printf("\n; end, put result on top of the stack\n");
      printf("define @mod_end_%d\n",label);
      printf("lda i 0 0\nind i\n");

      printf("\n; end of modulo %d\n", label);

      break;

    case AT_ILIST:
      pcodeGenValue(tree->left,s);
      pcodeGenValue(tree->right,s);
      break;

    default:
      fprintf(stderr,"KO\n");
      printf("ERROR : unrecognized type=%d in pcodeGenValue(..)\n", tree->type);
    }

    return 0;
  }
```