Université de Lorraine

RAPPORT DE PROJET : COMPILATION

SUJET

Ecriture d'un compilateur

Réalisé Par :

- BOURAIMA AFDAL
- EGLA JOSIANE
- ZOGBEMA YAHSE
- DOTSU OLYMPE

Encadré Par :

- Azim Roussanaly

Table des matières

Exemple 1 :

Code source :

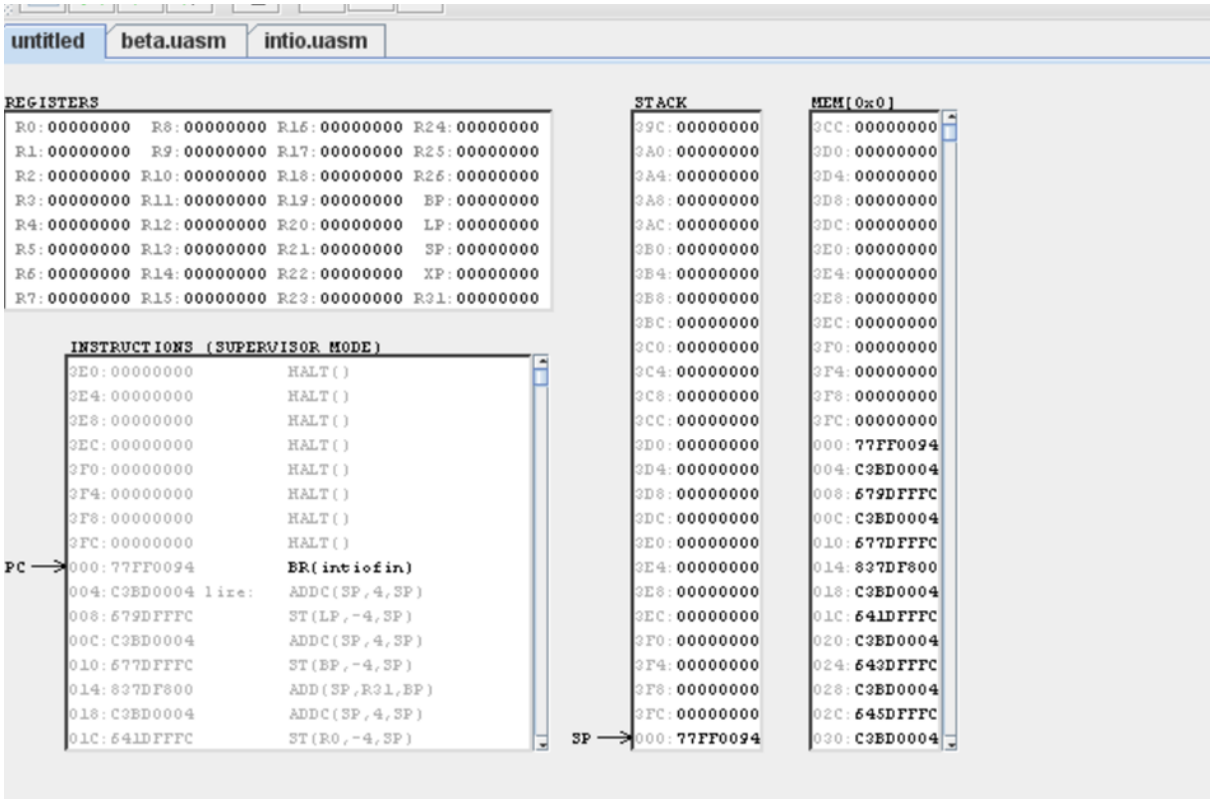
vide main () {}

code assembleur

```
.include beta.uasm

.include intio.uasm

.options tty


CMOVE(pile,SP)

BR(debut)

debut:

        CALL(main)

        HALT()

main:

        PUSH(LP)

        PUSH(BP)

        MOVE(SP,BP)

        ALLOCATE(0)

        DEALLOCATE(0)

        POP(BP)

        POP(LP)

        RTN()
```

Pile :

Exécution Bsim



TDS généré

```
{ nom: main type: INT cat: FONCTION nbparam: 0 nbloc: 1 }
```

Arbre généré

```
PROG
└─FONCTION/main
```

Code généré :

```
{ nom: main type: INT cat: FONCTION nbparam: 0 nbloc: 1 }

PROG
└─FONCTION/main

.include beta.uasm
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
debut:
    CALL(main)
    HALT()
main:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(0)
    DEALLOCATE(0)
    POP(BP)
    POP(LP)
    RTN()

pile:
```

Exemple 2 :

Code source

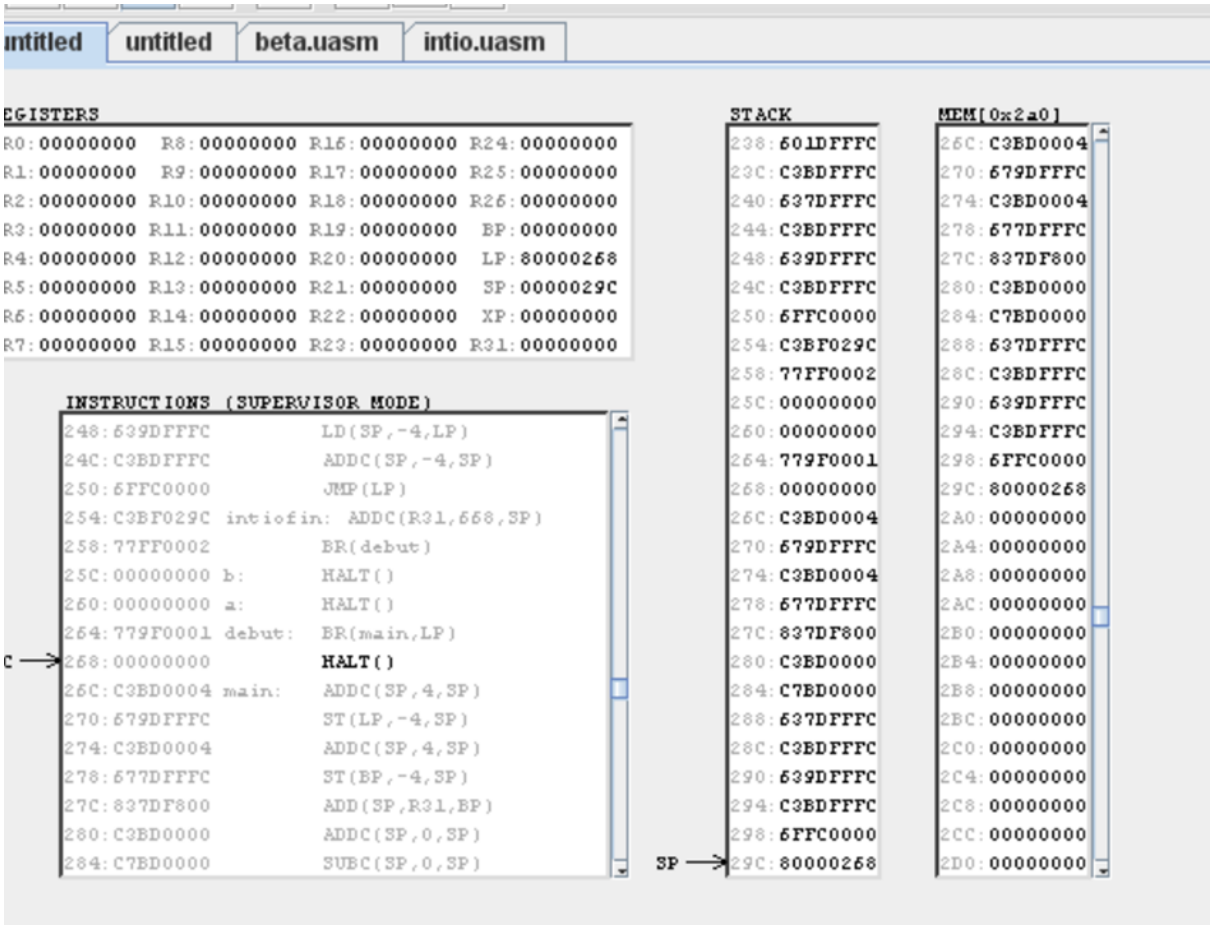
```
entier a; // variables globales
entier b;
void main () {}
```

Code assembleur :

```
 include beta.uasm

.include intio.uasm

.options tty


CMOVE(pile,SP)

BR(debut)
```

```
b:LONG(0)
a:LONG(0)
debut:
        CALL(main)
        HALT()
main:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(0)
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()

pile:
```

Exécution dans Bsim



TDS généré

```
{ nom: main type: INT cat: FONCTION nbparam: 0 nbloc: 1 }
{ nom: b type: INT cat: GLOBAL }
{ nom: a type: INT cat: GLOBAL }
```

Arbre généré

```
PROG
└─FONCTION/main
```

Code généré

```
.include beta.uasm
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
b:LONG(0)
a:LONG(0)
debut:
    CALL(main)
    HALT()
main:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(0)
    DEALLOCATE(0)
    POP(BP)
    POP(LP)
    RTN()

pile:
```

Exemple 3 :

Code source

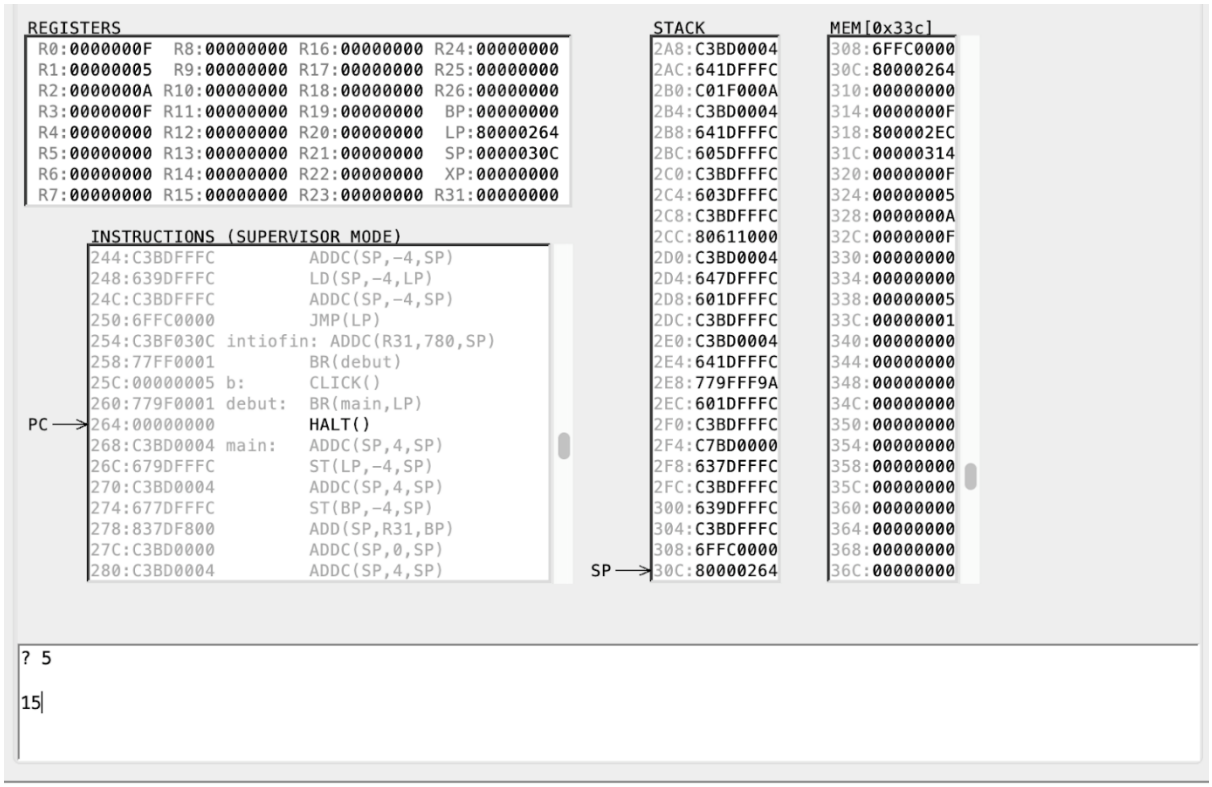
entier a;
entier b;

```
vide main()
{
        a=2;    //affectation
        b= lire(); //lecture
        ecrire (a); //ecriture
}
```

Code assembleur :

```
.include beta.uasm
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
b:LONG(0)
debut:
        CALL(main)
        HALT()
main:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(0)
        RDINT()
        PUSH(R0)
        POP(R0)
        ST(R0,b)
        LD(b,R0)
        PUSH(R0)
        CMOVE(10,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        ADD(R1,R2,R3)
        PUSH(R3)
        POP(R0)
        WRINT()
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()

pile:
```

Exécution dans Bsim



TDS généré

```
{ nom: main type: VOID cat: FONCTION nbparam: 0 nbloc: 1 }
{ nom: b type: INT cat: GLOBAL }
```

Arbre généré

```
PROG
└FONCTION/main
  └AFF
    └IDF/b
    └LIRE
  └ECR
    └PLUS
      └IDF/b
      └CONST/10
```

Code généré

```
.include beta.uasm
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
b:LONG(0)
debut:
    CALL(main)
    HALT()
main:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(0)
    RDINT()
    PUSH(R0)
    POP(R0)
    ST(R0,b)
    LD(b,R0)
    PUSH(R0)
    CMOVE(10,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
    ADD(R1,R2,R3)
    PUSH(R3)
    POP(R0)
    WRINT()
    DEALLOCATE(0)
    POP(BP)
    POP(LP)
    RTN()

pile:
```

Exemple 4

Code source

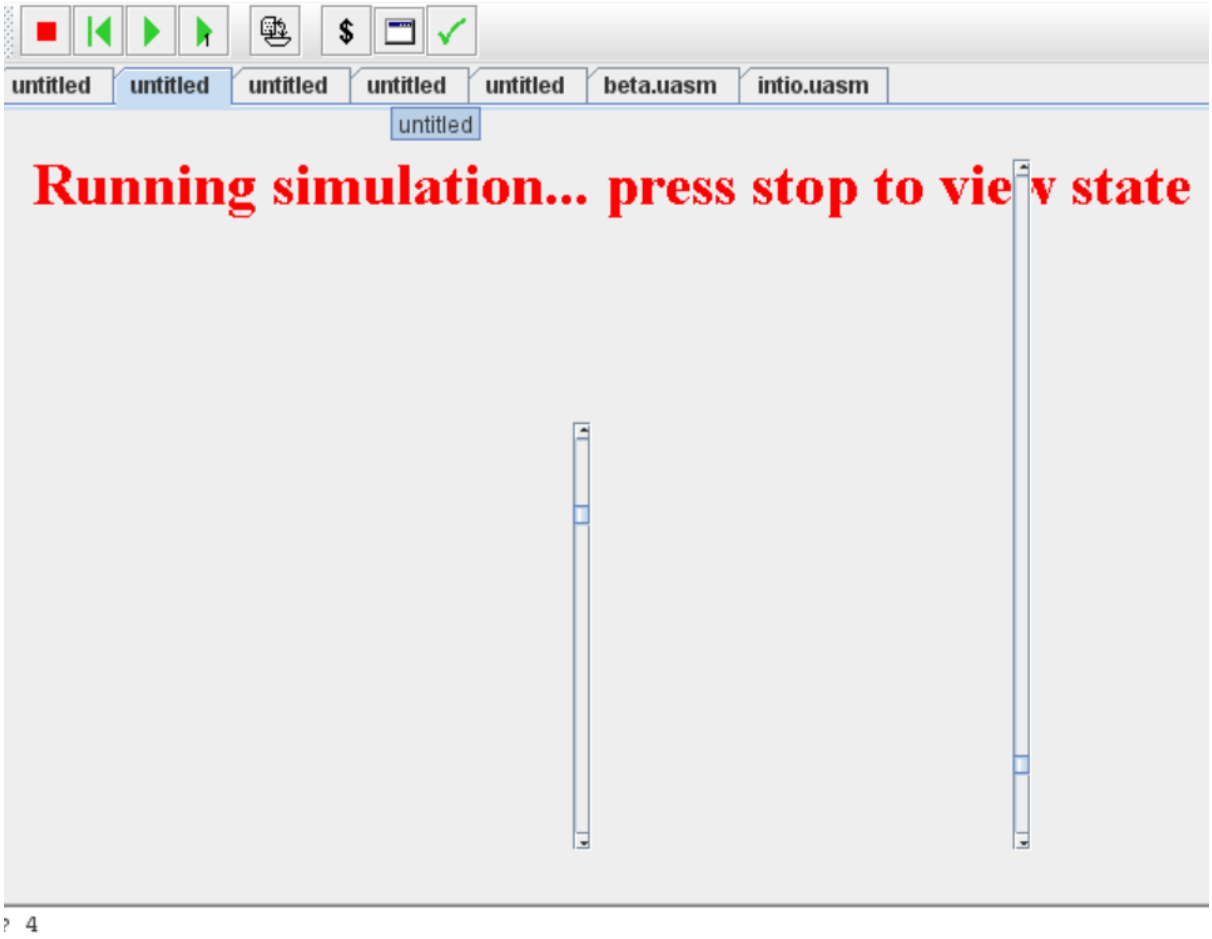
```
entier a;
entier b;
vide main() {
  a=lire ();
  b=5;
  si (a > b)
    {ecrire (a);}
```

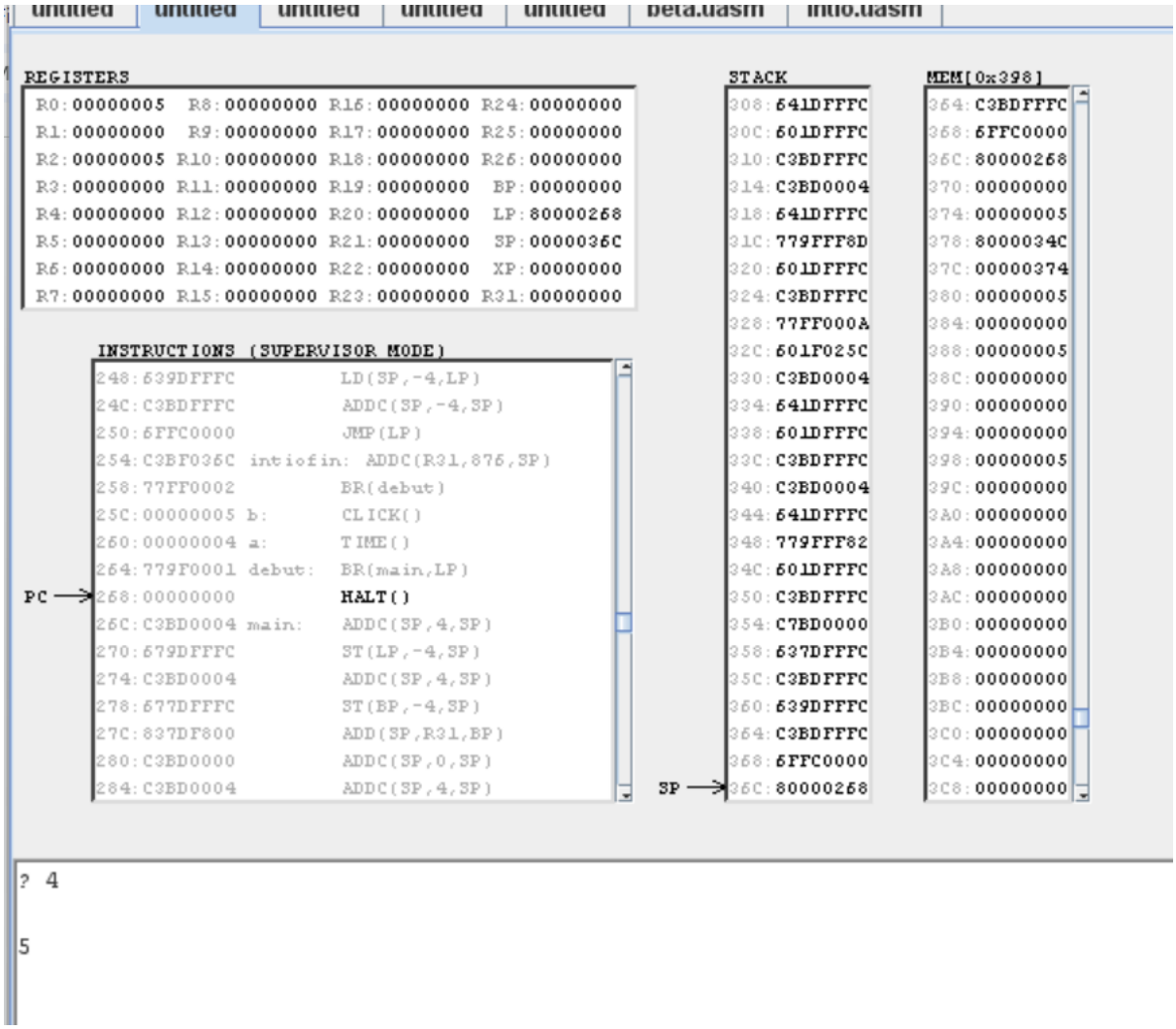
```
   sinon      {ecrire(b);}
}
```

Code assembleur :

```
.include beta.uasm
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
b:LONG(0)
a:LONG(0)
debut:
        CALL(main)
        HALT()
main:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(0)
        RDINT()
        PUSH(R0)
        POP(R0)
        ST(R0,a)
        CMOVE(5,R0)
        PUSH(R0)
        POP(R0)
        ST(R0,b)
        LD(a,R1)
        PUSH(R1)
        LD(b,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        CMPLT(R2,R1,R3)
        PUSH(R3)
        POP(R1)
        BF(R1,else)
        LD(a,R0)
        PUSH(R0)
        POP(R0)
        WRINT()
        BR(end_if)
        else:
        LD(b,R0)
        PUSH(R0)
        POP(R0)
        WRINT()
        end_if:
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()

pile:
```

Exécution dans Bsim





TDS généré

```
{ nom: main type: VOID cat: FONCTION nbparam: 0 nbloc: 1 }
{ nom: b type: INT cat: GLOBAL }
{ nom: a type: INT cat: GLOBAL }
```

Arbre généré

```
PROG
└─FONCTION/main
  └─AFF
    └─IDF/a
    └─LIRE
  └─AFF
    └─IDF/b
    └─CONST/5
  └─SI/0
    └─SUP
      └─IDF/a
      └─IDF/b
    └─BLOC
      └─ECR
        └─IDF/a
    └─BLOC
      └─ECR
        └─IDF/b
```

Code généré

```
.include beta.uasm
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
b:LONG(0)
a:LONG(0)
debut:
    CALL(main)
    HALT()
main:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(0)
    RDINT()
    PUSH(R0)
    POP(R0)
    ST(R0,a)
    CMOVE(5,R0)
    PUSH(R0)
    POP(R0)
    ST(R0,b)
    LD(a,R1)
    PUSH(R1)
    LD(b,R0)
    PUSH(R0)
```

```
    POP(R2)
    POP(R1)
    CMPLT(R2,R1,R3)
    PUSH(R3)
    POP(R1)
    BF(R1,else)
    LD(a,R0)
    PUSH(R0)
    POP(R0)
    WRINT()
    BR(end_if)
    else:
    LD(b,R0)
    PUSH(R0)
    POP(R0)
    WRINT()
    end_if:
    DEALLOCATE(0)
    POP(BP)
    POP(LP)
    RTN()

pile:
```

Exemple 5

Code source

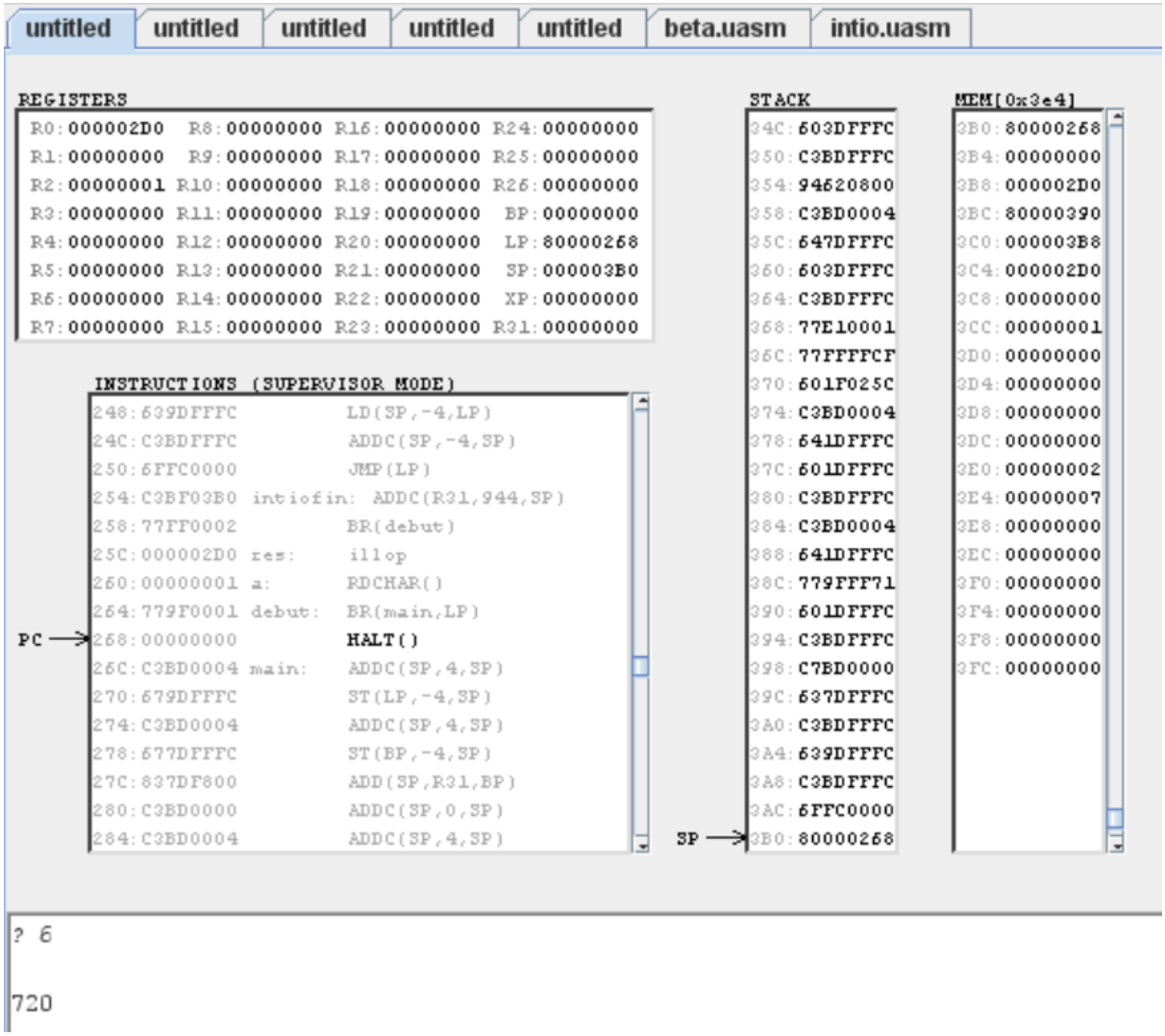
```
entier a;
entier res=1;
vide main (){
        a=lire();
        tantque(a>1) {
                res=res*a;
                a=a-1;
        }
   ecrire(res);
}
```

Code assembleur :

```
.include beta.uasm
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
res:LONG(1)
a:LONG(0)
debut:
        CALL(main)
        HALT()
main:
        PUSH(LP)
        PUSH(BP)
```

```
        MOVE(SP,BP)
        ALLOCATE(0)
        RDINT()
        PUSH(R0)
        POP(R0)
        ST(R0,a)
        BR(condition_check)
        loop:
        LD(res,R0)
        PUSH(R0)
        LD(a,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        MUL(R1,R2,R3)
        PUSH(R3)
        POP(R0)
        ST(R0,res)
        LD(a,R0)
        PUSH(R0)
        CMOVE(1,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        SUB(R1,R2,R3)
        PUSH(R3)
        POP(R0)
        ST(R0,a)
        condition_check:
        LD(a,R1)
        PUSH(R1)
        CMOVE(1,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        CMPLT(R2,R1,R3)
        PUSH(R3)
        POP(R1)
        BF(R1,loop_exit)
        BR(loop)
        loop_exit:
        LD(res,R0)
        PUSH(R0)
        POP(R0)
        WRINT()
        DEALLOCATE(0)
        POP(BP)
        POP(LP)
        RTN()

pile:
```

Exécution dans Bsim



TDS généré

```
{ nom: res type: INT cat: LOCAL val: 1 rang: 0 scope: main }
{ nom: main type: VOID cat: FONCTION nbparam: 0 nbloc: 1 }
{ nom: a type: INT cat: GLOBAL }
```

Arbre généré :

```
PROG
└─FONCTION/main
  └─AFF
    └─IDF/a
    └─LIRE
  └─TQ/0
    └─SUP
      └─IDF/a
      └─CONST/1
    └─BLOC
      └─AFF
        └─IDF/res
        └─MUL
          └─IDF/res
          └─IDF/a
      └─AFF
        └─IDF/a
        └─MOINS
          └─IDF/a
          └─CONST/1
  └─ECR
    └─IDF/res
```

Code généré

```
.include beta.uasm
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
a:LONG(0)
debut:
    CALL(main)
    HALT()
main:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(1)
    RDINT()
    PUSH(R0)
    POP(R0)
    ST(R0,a)
    BR(condition_check)
    loop:
    GETFRAME(0,R0)
    PUSH(R0)
    LD(a,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
    MUL(R1,R2,R3)
```

```
    PUSH(R3)
    POP(R0)
    PUTFRAME(R0,0)
    LD(a,R0)
    PUSH(R0)
    CMOVE(1,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
    SUB(R1,R2,R3)
    PUSH(R3)
    POP(R0)
    ST(R0,a)
    condition_check:
    LD(a,R1)
    PUSH(R1)
    CMOVE(1,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
    CMPLT(R2,R1,R3)
    PUSH(R3)
    POP(R1)
    BF(R1,loop_exit)
    BR(loop)
    loop_exit:
    GETFRAME(0,R0)
    PUSH(R0)
```

```
    POP(R0)
    WRINT()
    DEALLOCATE(1)
    POP(BP)
    POP(LP)
    RTN()

pile:
```

Exemple 6

Code source

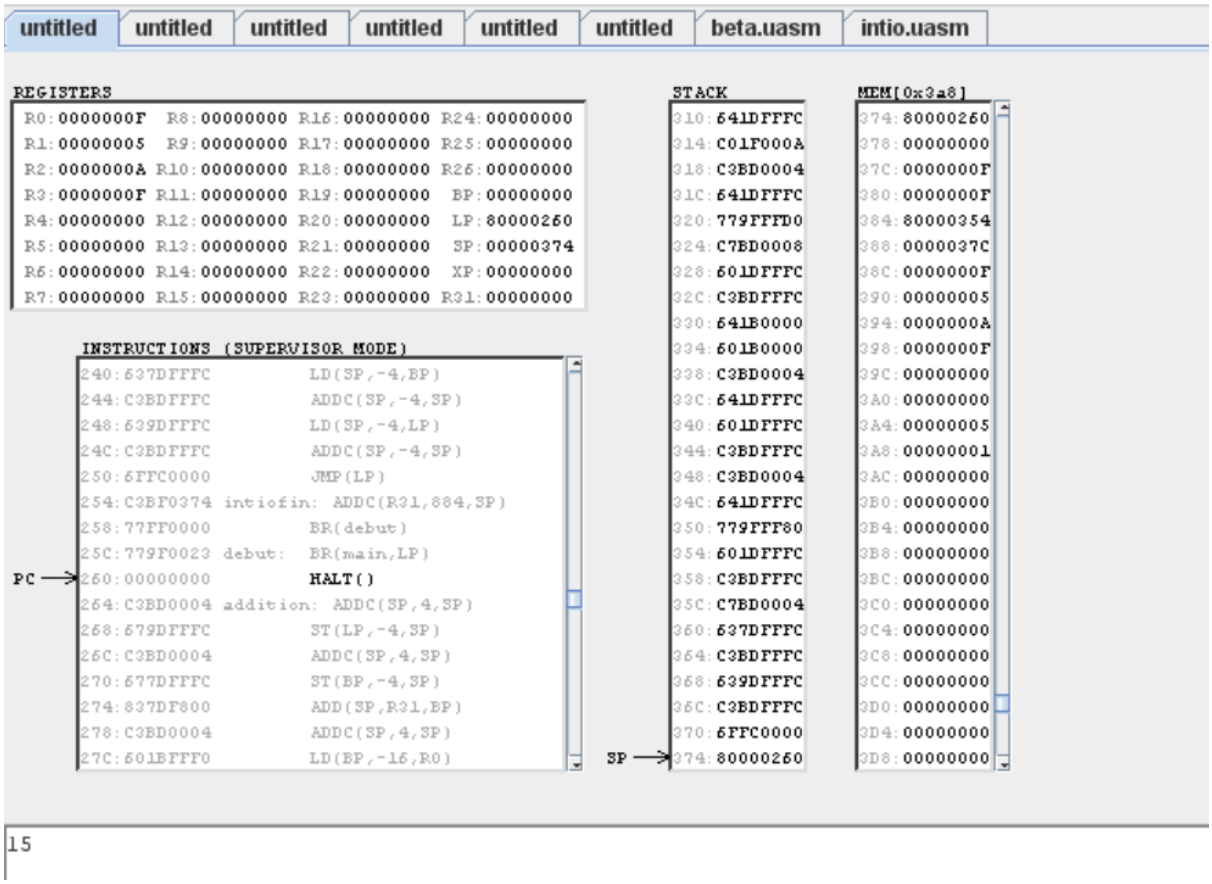
```
entier addition (entier a, entier b){
        entier S;
        S=a+b;
        retourne S;}
vide main () {
        entier res;
        res = addition (5,10);
        ecrire (res);}
```

Code assembleur

```
.include beta.uasm
```

```
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
debut:
        CALL(main)
        HALT()
addition:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(1)
        GETFRAME(-16,R0)
        PUSH(R0)
        GETFRAME(-12,R0)
        PUSH(R0)
        POP(R2)
        POP(R1)
        ADD(R1,R2,R3)
        PUSH(R3)
        POP(R0)
        PUTFRAME(R0,0)
        GETFRAME(0,R0)
        PUSH(R0)
        POP(R0)
        PUTFRAME(R0,-20)
        DEALLOCATE(1)
        POP(BP)
        POP(LP)
        RTN()
main:
        PUSH(LP)
        PUSH(BP)
        MOVE(SP,BP)
        ALLOCATE(1)
        ALLOCATE(1)
        CMOVE(5,R0)
        PUSH(R0)
        CMOVE(10,R0)
        PUSH(R0)
        CALL(addition)
        DEALLOCATE(2)
        POP(R0)
        PUTFRAME(R0,0)
        GETFRAME(0,R0)
        PUSH(R0)
        POP(R0)
        WRINT()
        DEALLOCATE(1)
        POP(BP)
        POP(LP)
        RTN()

pile:
```

Exécution dans Bsim



TDS généré

```
{ nom: a type: INT cat: PARAM rang: 0 scope: addition }
{ nom: res type: INT cat: LOCAL rang: 0 scope: main }
{ nom: addition type: INT cat: FONCTION nbparam: 2 nbloc: 1 }
{ nom: main type: VOID cat: FONCTION nbparam: 0 nbloc: 1 }
{ nom: b type: INT cat: PARAM rang: 1 scope: addition }
{ nom: S type: INT cat: LOCAL rang: 0 scope: addition }
```

Arbre généré

```
PROG
└FONCTION/addition
  └AFF
    └IDF/S
    └PLUS
      └IDF/a
      └IDF/b
  └RET/FONCTION/addition
    └IDF/S
└FONCTION/main
  └AFF
    └IDF/res
    └APPEL/FONCTION/addition
      └CONST/5
      └CONST/10
  └ECR
    └IDF/res
```

Code généré :

```
.include beta.uasm
.include intio.uasm
.options tty

CMOVE(pile,SP)
BR(debut)
debut:
    CALL(main)
    HALT()
addition:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(1)
    GETFRAME(-16,R0)
    PUSH(R0)
    GETFRAME(-12,R0)
    PUSH(R0)
    POP(R2)
    POP(R1)
    ADD(R1,R2,R3)
    PUSH(R3)
    POP(R0)
    PUTFRAME(R0,0)
    GETFRAME(0,R0)
    PUSH(R0)
    POP(R0)
    PUTFRAME(R0,-20)
    DEALLOCATE(1)
    POP(BP)
    POP(LP)
    RTN()
```

```
main:
    PUSH(LP)
    PUSH(BP)
    MOVE(SP,BP)
    ALLOCATE(1)
    ALLOCATE(1)
    CMOVE(5,R0)
    PUSH(R0)
    CMOVE(10,R0)
    PUSH(R0)
    CALL(addition)
    DEALLOCATE(2)
    POP(R0)
    PUTFRAME(R0,0)
    GETFRAME(0,R0)
    PUSH(R0)
    POP(R0)
    WRINT()
    DEALLOCATE(1)
    POP(BP)
    POP(LP)
    RTN()

pile:
```

Exemple 7

Code source

```
int factorielle (int n) {  // récursif
        If (n=0){
                return 1;
        }
        return n * factorielle (n-1);
}

entier factorielle (entier n) {
```

```
        Si (n=0){
                retourne 1;
        }
        retourne n * factorielle (n-1);
}
```

Grammaire CUP du langage

```
/*Grammaire CUP du projet Expression*/

package generated.fr.ul.miage.rs.langage;

import fr.ul.miage.arbre.*;

import fr.ul.miage.us.Tds;

import fr.ul.miage.us.Item;

import fr.ul.miage.us.Cat;

import fr.ul.miage.us.Type;

import java.util.ArrayList;

import java.util.List;

parser code{:

        //public Noeud res = null;

        //public ArrayList res = new ArrayList<Object>();

  //public ArrayList<ArrayList<Object>> tuples = new ArrayList<ArrayList<Object>>(); // L'objet qu'on retourne Ã  la fin contenant l'arbre et la tds

  //public Tds fonction = new Tds();

:}

/*-------------------------------------------*/

/* la grammaire */

/* 1) terminaux */

terminal VIRGULE,PVIRGULE,ADD, MUL,SOUS,DIV, PO,
PF,SI,SINON,ECRIRE,LIRE,TANTQUE,RETOURNE,VIDE,ENTIER,IDF,SUP,INF,SUPE,INFE,EG,DIF
F,AO,AF,AFFE;

terminal Integer NUM;
```

```
/* 2) non terminaux */

non terminal langage;

non terminal Noeud
repetitive,bloc,condition,alternative,appel,expression,param,affectation,instruction,facteur,at
ome;

non terminal ArrayList
lparam,l_instructions,l_param_declare,l_declaration_locale,l_declaration_globale,function,l_fu
nction,prog;

non terminal Item declaration_globale,declaration_locale,param_declare;

/* 3) Axiome/Start */

start with langage;

/*-----------------------------------------------*/

/*4) regles de production */

langage ::=prog {:;:}

;

// Un programme est une liste de fonction

prog ::= l_declaration_globale l_function {:;:}

     |l_function:lf       {:;:}

;

// Liste de fonctions

l_function ::=  function l_function  {:;:}

          |function:f   {:;:}

;


function ::= VIDE IDF PO l_param_declare PF AO l_declaration_locale l_instructions AF  {:;:}

        |ENTIER IDF PO l_param_declare PF AO l_declaration_locale l_instructions RETOURNE
expression PVIRGULE AF  {:;:}

        // Une fonction peut avoir un bloc vide

        |VIDE IDF:idf PO l_param_declare:lp PF AO  AF     {:;:}


        |VIDE IDF PO l_param_declare PF AO l_instructions AF  {:;:}

;

// liste de paramÃ¨tres dÃ©clarÃ©s dans une fonction
```

```
l_param_declare ::=l_param_declare:lp VIRGULE param_declare:p   {:;:}
            |param_declare:p   {:;:}
            |             {:;:}
;
// Un paramÃ¨tre dÃ©clarÃ© dans une fonction
param_declare ::= ENTIER IDF:idf   {:;:}
;

// Liste de dÃ©claration locale
l_declaration_locale ::=l_declaration_locale declaration_locale
            |declaration_locale   {:;:}
;

// declaration locale
declaration_locale ::= ENTIER IDF PVIRGULE {:;:}
              |ENTIER IDF AFFE NUM PVIRGULE {:;:}
;
// liste de declaration globale
l_declaration_globale ::=l_declaration_globale  declaration_globale:   {:;:}
            |declaration_globale   {:;:}
;

// Declaration globale avant la creation de fonctions
declaration_globale ::= ENTIER IDF PVIRGULE {:;:}
          |ENTIER IDF AFFE NUM PVIRGULE {:;:}
;
//Repetitive
repetitive ::= TANTQUE PO condition PF bloc {:;:}
;
//Alternative
alternative ::= SI PO condition:cond PF bloc:bloc_alors {:;:}
```

```
            |SI PO condition PF bloc SINON bloc  {:;:}
;
//Condition
condition ::= expression EG expression    {:;:}
          |expression INF expression    {:;:}
          |expression INFE expression   {:;:}
          |expression SUP expression    {:;:}
          |expression SUPE expression   {:;:}
          |expression DIFF expression   {:;:}
;
//Bloc
bloc ::=AO  l_instructions AF  {:;:}
;
l_instructions::= instruction l_instructions   {:;:}
             |instruction                {:;:}
;

//Instruction
instruction ::= ECRIRE expression PVIRGULE  {:;:}
             |affectation PVIRGULE {:;:}
             |alternative          {:;:}
             |repetitive           {:;:}

;
//Affectation c'est un idf = expression
affectation ::= IDF AFFE expression   {:;:}

          | IDF AFFE LIRE PO PF     {:;:}
;
//Un appel c'est :idf ( liste de paramÃ¨tres )
appel ::= IDF PO lparam PF              {:;:}
```

```
;
lparam ::= param VIRGULE lparam:  {:;:}
      |param   {:;:}
;
param ::=expression:x {:;:}
;
expression ::= expression ADD facteur          {:;:}
   |expression SOUS facteur    {:;:}
            | facteur                                                    {:;:}
            ;
facteur ::= facteur MUL atome           {:;:}
   |facteur DIV atome   {:;:}
            | atome                                          {:;:}
            ;
atome ::= NUM                                   {:;:}
   |IDF           {:;:}
            | PO expression PF      {:;:}
   | appel          {:;:}
            ;
```

Grammaire enrichie du langage

```
/*Grammaire CUP du projet Expression*/
package generated.fr.ul.miage.rs.langage;
import fr.ul.miage.arbre.*;
import fr.ul.miage.us.Tds;
import fr.ul.miage.us.Item;
import fr.ul.miage.us.Cat;
import fr.ul.miage.us.Type;
import java.util.ArrayList;
import java.util.List;
parser code{:
```

```
        //public Noeud res = null;

        public ArrayList res = new ArrayList<Object>();

   public ArrayList<ArrayList<Object>> tuples = new ArrayList<ArrayList<Object>>(); // L'objet qu'on retourne à la fin contenant l'arbre et la tds

   public Tds fonction = new Tds();

:}

/*-------------------------------------------*/

/* la grammaire */

/* 1) terminaux */

terminal VIRGULE,PVIRGULE,ADD, MUL,SOUS,DIV, PO, PF,SI,SINON,ECRIRE,LIRE,TANTQUE,RETOURNE,VIDE,ENTIER,IDF,SUP,INF,SUPE,INFE,EG,DIFF,AO,AF,AFFE;

terminal Integer NUM;

/* 2) non terminaux */

non terminal langage;

non terminal Noeud repetitive,bloc,condition,alternative,appel,expression,param,affectation,instruction,facteur,atome;

non terminal ArrayList lparam,l_instructions,l_param_declare,l_declaration_locale,l_declaration_globale,function,l_function,prog;

non terminal Item declaration_globale,declaration_locale,param_declare;

/* 3) Axiome/Start */

start with langage;

/*-------------------------------------------*/

/*4) regles de production */

langage ::=prog:p {: res= (ArrayList<Object>)p;:}

//langage::=function:f {:tuple.addAll(f);:}

;

// Un programme est une liste de fonction

prog ::= l_declaration_globale:ld l_function:lf {:RESULT = new ArrayList<Object>();

            // nous alons fusionner tout les items crée jusque là dans une liste

            ArrayList items = new ArrayList<Item>();

            // et toutes les fonctions dans une autre liste
```

```
                ArrayList fonctions = new ArrayList<Fonction>();
                for(ArrayList<Object> tuple:(ArrayList<ArrayList<Object>>)lf){
                items.addAll((ArrayList<Item>)(tuple.get(0)));
                fonctions.add((tuple.get(1)));}
                // On crée la tds à partir des items et on la met dans notre resultat
                Tds tds =new Tds();
                //On ajoute les variables globales
                for (Item item:(ArrayList<Item>)ld){tds.addItem(item);}
                // On rajoute les fonctions et les variables locales
                for (Item item:(ArrayList<Item>)items){tds.addItem(item);}
                RESULT.add(tds);
                // On cree le noeud prog et on lui lie tous les noeuds fonctions
                Prog prog = new Prog();
                for (Fonction
fonction:(ArrayList<Fonction>)fonctions){prog.ajouterUnFils(fonction);}
                RESULT.add(prog);:}
    |l_function:lf      {:RESULT = new ArrayList<Object>();
                    // nous alons fusionner tout les items crée jusque là dans une liste
                    ArrayList items = new ArrayList<Item>();
                    // et toutes les fonctions dans une autre liste
                    ArrayList fonctions = new ArrayList<Fonction>();
                    for(ArrayList<Object> tuple:(ArrayList<ArrayList<Object>>)lf){
                    items.addAll((ArrayList<Item>)(tuple.get(0)));
                    fonctions.add((tuple.get(1)));}
                    // On crée la tds à partir des items et on la met dans notre resultat
                    Tds tds =new Tds();
                    // On rajoute les fonctions et les variables locales
                    for (Item item:(ArrayList<Item>)items){tds.addItem(item);}
                    RESULT.add(tds);
                    // On cree le noeud prog et on lui lie tous les noeuds fonctions
                    Prog prog = new Prog();
```

```
                    for (Fonction
fonction:(ArrayList<Fonction>)fonctions){prog.ajouterUnFils(fonction);}

                    RESULT.add(prog);:}

;

// Liste de fonctions

l_function ::=  function:f l_function:lf  {:System.out.println("liste de fonction");

                        RESULT = lf;

                        RESULT.add(0,f);:}

        |function:f   {:RESULT = new ArrayList<ArrayList<Object>>();

                    RESULT.add(f);:}

;


function ::= VIDE IDF:idf PO l_param_declare:lp PF AO l_declaration_locale:dl l_instructions:li
AF  {: System.out.println("fonction");

                                RESULT = new ArrayList<Object>() ;

                                // On rajoute des informations sur la fonction à la tds

                                ArrayList matds = new ArrayList<Item>();

                                matds.add(new
Item((String)idf,Type.VOID,Cat.FONCTION,((ArrayList<Item>)lp).size(),1));

                                // On defini les scope de nos paramètres et variable locale comme
étant la fonction

                                for (Item
item:(ArrayList<Item>)lp){((Item)item).setScope((Item)(matds.get(0)));}

                                for (Item
item:(ArrayList<Item>)dl){((Item)item).setScope((Item)(matds.get(0)));}

                                matds.addAll(lp);

                                matds.addAll(dl);

                                RESULT.add(matds);

                                Fonction fonc = new Fonction((String)idf);

                                fonc.setFils(li);

                                RESULT.add(fonc);:}

      |ENTIER IDF:idf PO l_param_declare:lp PF AO l_declaration_locale:dl l_instructions:li
RETOURNE expression:x PVIRGULE AF  {: System.out.println("fonction");

                                        RESULT = new ArrayList<Object>() ;
```

```
                                        // On rajoute des informations sur la fonction à la tds
                                        ArrayList matds = new ArrayList<Item>();
                                        matds.add(new
Item((String)idf,Type.INT,Cat.FONCTION,((ArrayList<Item>)lp).size(),1));
                                        // On defini les scope de nos paramètres et variable locale
comme étant la fonction
                                        for (Item
item:(ArrayList<Item>)lp){((Item)item).setScope((Item)(matds.get(0)));}
                                        for (Item
item:(ArrayList<Item>)dl){((Item)item).setScope((Item)(matds.get(0)));}
                                        matds.addAll(lp);
                                        matds.addAll(dl);
                                        RESULT.add(matds);
                                        Fonction fonc = new Fonction((String)idf);
                                        fonc.setFils(li);
                                       // ajouter le retour de la fonction
                                        Retour retour =new Retour(fonc);
                                        retour.setLeFils(x);
                                        fonc.ajouterUnFils(retour);
                                        RESULT.add(fonc);


                                       :}
      // Une fonction peut avoir un bloc vide
      |VIDE IDF:idf PO l_param_declare:lp PF AO   AF      {:RESULT = new ArrayList<Object>() ;
                                    // On rajoute des informations sur la fonction à la tds
                                     ArrayList matds = new ArrayList<Item>();
                                     matds.add(new
Item((String)idf,Type.INT,Cat.FONCTION,((ArrayList<Item>)lp).size(),0));
                                     Fonction fonc = new Fonction((String)idf);
                                     RESULT.add(matds);
                                     RESULT.add(fonc);:}
```

```
        |VIDE IDF:idf PO l_param_declare:lp PF AO l_instructions:li AF   {:RESULT = new
ArrayList<Object>() ;
                                                 // On rajoute des informations sur la fonction à la tds
                                                ArrayList matds = new ArrayList<Item>();
                                                matds.add(new
Item((String)idf,Type.VOID,Cat.FONCTION,((ArrayList<Item>)lp).size(),1));
                                                // On defini les scope de nos paramètres et variable locale
comme étant la fonction
                                                for (Item
item:(ArrayList<Item>)lp){((Item)item).setScope((Item)(matds.get(0)));}
                                                matds.addAll(lp);
                                                RESULT.add(matds);
                                                Fonction fonc = new Fonction((String)idf);
                                                fonc.setFils(li);
                                                RESULT.add(fonc);:}
;
l_param_declare ::=l_param_declare:lp VIRGULE param_declare:p    {:RESULT = lp;
                                        p.setRang(RESULT.size()); // le rang du paramètre sera la taille
actuelle de la liste
                                        RESULT.add(0,p);:}
            |param_declare:p    {:RESULT = new ArrayList<Item>();
                        p.setRang(0);
                        RESULT.add(p);:}
            |               {:RESULT = new ArrayList<Item>();:}
;

param_declare ::= ENTIER IDF:idf    {:RESULT = new Item((String)idf,Type.INT,Cat.PARAM);:}
;

l_declaration_locale ::=l_declaration_locale:ld declaration_locale:d    {:RESULT = ld;
                                        d.setRang(RESULT.size()); // le rang de la variable locale sera la
taille actuelle de la liste
                                        RESULT.add(0,d);
```

```
                        :}
            |declaration_locale:d    {:RESULT = new ArrayList<Item>();
                        d.setRang(0);
                        RESULT.add(d);:}
;

// declaration locale
declaration_locale ::= ENTIER IDF:idf PVIRGULE {:RESULT = new
Item((String)idf,Type.INT,Cat.LOCAL);:}
              |ENTIER IDF:idf AFFE NUM:val PVIRGULE {:RESULT = new
Item((String)idf,Type.INT,Cat.LOCAL,val);:}
;
// liste de declaration globale
l_declaration_globale ::=l_declaration_globale:ld  declaration_globale:d   {:RESULT = ld;

                                    RESULT.add(0,d);
                        :}
            |declaration_globale:d    {:RESULT = new ArrayList<Item>();

                        RESULT.add(d);:}
;

// Declaration globale avant la creation de fonctions
declaration_globale ::= ENTIER IDF:idf PVIRGULE {:RESULT = new
Item((String)idf,Type.INT,Cat.GLOBAL);
                              :}
          |ENTIER IDF:idf AFFE NUM:val PVIRGULE {:RESULT = new
Item((String)idf,Type.INT,Cat.GLOBAL,val);
                                :}
;
//Repetitive
repetitive ::= TANTQUE PO condition:cond PF bloc:bloc_faire {:System.out.println("repetitive");
```

```
                                    RESULT = new TantQue();
                                    ((TantQue)RESULT).setCondition(cond);
                                    ((TantQue)RESULT).setBloc((Bloc)bloc_faire);:}
;
//Alternative
alternative ::= SI PO condition:cond PF bloc:bloc_alors {:System.out.println("alternative");
                        RESULT = new Si();
                        ((Si)RESULT).setCondition(cond);
                        ((Si)RESULT).setBlocAlors((Bloc)bloc_alors);:}
        |SI PO condition:cond PF bloc:bloc_alors SINON bloc:bloc_sinon  {:RESULT = new Si();
                            ((Si)RESULT).setCondition(cond);
                            ((Si)RESULT).setBlocAlors((Bloc)bloc_alors);
                            ((Si)RESULT).setBlocSinon((Bloc)bloc_sinon);:}
;
//Condition
condition ::= expression:e1 EG expression:e2    {:System.out.println("Condition");
                            RESULT = new Egal();
                            ((Egal)RESULT).setFilsGauche(e1);
                            ((Egal)RESULT).setFilsDroit(e2);:}
        |expression:e1 INF expression:e2    {:RESULT = new Inferieur();
                            ((Inferieur)RESULT).setFilsGauche(e1);
                            ((Inferieur)RESULT).setFilsDroit(e2);:}
        |expression:e1 INFE expression:e2   {:RESULT = new InferieurEgal();
                            ((InferieurEgal)RESULT).setFilsGauche(e1);
                            ((InferieurEgal)RESULT).setFilsDroit(e2);:}
        |expression:e1 SUP expression:e2    {:RESULT = new Superieur();
                            ((Superieur)RESULT).setFilsGauche(e1);
                            ((Superieur)RESULT).setFilsDroit(e2);:}
        |expression:e1 SUPE expression:e2   {:RESULT = new SuperieurEgal();
                            ((SuperieurEgal)RESULT).setFilsGauche(e1);
                            ((SuperieurEgal)RESULT).setFilsDroit(e2);:}
```

```
        |expression:e1 DIFF expression:e2   {:RESULT = new Different();
                                    ((Different)RESULT).setFilsGauche(e1);
                                    ((Different)RESULT).setFilsDroit(e2);:}
;
//Bloc
bloc ::=AO  l_instructions:li AF   {:RESULT = new Bloc();
                        ((Bloc)RESULT).setFils(li);:}
;
l_instructions::= instruction:i l_instructions:li    {:RESULT = li;
                                    RESULT.add(0,i);:}
           |instruction:i                {:RESULT = new ArrayList<Noeud>();
                                    RESULT.add(i);:}
;

//Instruction
instruction ::= ECRIRE expression:x PVIRGULE  {:    RESULT = new Ecrire();
                                    ((Ecrire)RESULT).setLeFils(x);:}
          |affectation:aff PVIRGULE {:RESULT=aff;:}
          |alternative:alt         {:RESULT = alt ;:}
          |repetitive:rep          {:RESULT = rep;:}
          //| IDF:idf AFFE appel:a        {:RESULT = new Affectation();
           //                              ((Affectation)RESULT).setFilsGauche(new Idf(idf));
            //                             ((Affectation)RESULT).setFilsDroit(a);:}

;
//Affectation c'est un idf = expression
affectation ::= IDF:idf AFFE expression:x   {:RESULT = new Affectation();
                            ((Affectation)RESULT).setFilsGauche(new Idf(idf));
                            ((Affectation)RESULT).setFilsDroit(x);:}

        | IDF:idf AFFE LIRE PO PF     {:RESULT = new Affectation();
```

```
                        ((Affectation)RESULT).setFilsGauche(new Idf(idf));
                        ((Affectation)RESULT).setFilsDroit(new Lire());:}
;
//Un appel c'est :idf ( liste de paramètres )
appel ::= IDF:idf PO lparam:lp PF          {:
                          RESULT = new Appel(new Fonction(idf));
                          ((Appel)RESULT).setFils(lp);:}

;
lparam ::= param:p VIRGULE lparam:lp  {:RESULT = lp;
                    RESULT.add(0,p);
                 :}
    |param:p   {:RESULT = new ArrayList<Noeud>();
              RESULT.add(p);:}
;
param ::=expression:x {:RESULT =x;:}
;
expression ::= expression:x ADD facteur:f     {:  System.out.println("Addition");
                          RESULT =new Plus();
                          ((Plus)RESULT).setFilsGauche(x);
                          ((Plus)RESULT).setFilsDroit(f);:}
  |expression:x SOUS facteur:f        {:  System.out.println("Soustraction");
                           RESULT =new Moins();
                          ((Moins)RESULT).setFilsGauche(x);
                          ((Moins)RESULT).setFilsDroit(f);:}
        | facteur:f                                                    {: RESULT = f; :}
        ;
facteur ::= facteur:f MUL atome:a      {: RESULT = new Multiplication();
                    ((Multiplication)RESULT).setFilsGauche(f);
                    ((Multiplication)RESULT).setFilsDroit(a);
                    System.out.println("Multiplication");:}
```

```
  |facteur:f DIV atome:a          {: RESULT = new Division();
                                ((Division)RESULT).setFilsGauche(f);
                                ((Division)RESULT).setFilsDroit(a);
                                System.out.println("Division");:}
        | atome:a                                                    {: RESULT = a; :}
        ;
atome ::= NUM:n                                    {: RESULT = new Const(n);
                        System.out.println("Constante");:}
  |IDF:idf            {:RESULT = new Idf(idf);:}
        | PO expression:x PF    {: RESULT = x; :}
  | appel:ap            {:RESULT =ap;:}
        ;
```

Bilan critique

Nous n'avons pas réussi à générer l'arbre et la tds pour l'exemple 7. Un problème dans notre grammaire que nous tentons toujours de résoudre.

Conclusion

Ce projet aura été challengeant et nous aura permis de comprendre tout le processus qui se cachait derrière la réalisation d'un langage de programmation viable. Et une fois qu'on l'a compris, cela semble si évident.