

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 : .....	2
Exemple 2 : .....	4
Exemple 3 : .....	7
Exemple 4 .....	10
Exemple 5 .....	15
Code source .....	15
Exemple 6 .....	20
Code source .....	20
Code assembleur .....	20
Exemple 7 .....	23
Grammaire CUP du langage .....	24
Grammaire enrichie du langage .....	28
Bilan critique .....	38

## 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*

untitled    beta.uasm    intio.uasm		
<b>REGISTERS</b>		
R0: 00000000	R8: 00000000	R16: 00000000
R1: 00000000	R9: 00000000	R17: 00000000
R2: 00000000	R10: 00000000	R18: 00000000
R3: 00000000	R11: 00000000	R19: 00000000
R4: 00000000	R12: 00000000	R20: 00000000
R5: 00000000	R13: 00000000	R21: 00000000
R6: 00000000	R14: 00000000	R22: 00000000
R7: 00000000	R15: 00000000	R23: 00000000
		R24: 00000000
		R25: 00000000
		R26: 00000000
		BP: 00000000
		LP: 00000000
		SP: 00000000
		XP: 00000000
		R31: 00000000
<b>INSTRUCTIONS (SUPERVISOR MODE)</b>		
0E0: 00000000	HALT()	
0E4: 00000000	HALT()	
0E8: 00000000	HALT()	
0EC: 00000000	HALT()	
0F0: 00000000	HALT()	
0F4: 00000000	HALT()	
0F8: 00000000	HALT()	
0FC: 00000000	HALT()	
PC → 000: 77FF0094	BR(intiofin)	
004: C3ED0004	line: ADDC(SP, 4, SP)	
008: 679DFFFC	ST(LP, -4, SP)	
00C: C3ED0004	ADDC(SP, 4, SP)	
010: 677DFFFC	ST(BP, -4, SP)	
014: 837DF800	ADD(SP, R31, BP)	
018: C3ED0004	ADDC(SP, 4, SP)	
01C: 641DFFFC	ST(R0, -4, SP)	
<b>STACK</b>		
09C: 00000000		
0A0: 00000000		
0A4: 00000000		
0A8: 00000000		
0AC: 00000000		
0B0: 00000000		
0B4: 00000000		
0B8: 00000000		
0BC: 00000000		
0C0: 00000000		
0C4: 00000000		
0C8: 00000000		
0CC: 00000000		
0D0: 00000000		
0D4: 00000000		
0D8: 00000000		
0DC: 00000000		
0E0: 00000000		
0E4: 00000000		
0E8: 00000000		
0EC: 00000000		
0F0: 00000000		
0F4: 00000000		
0F8: 00000000		
0FC: 00000000		
SP → 000: 77FF0094		
<b>MEM[0x0]</b>		
0CC: 00000000		
0D0: 00000000		
0D4: 00000000		
0D8: 00000000		
0DC: 00000000		
0E0: 00000000		
0E4: 00000000		
0E8: 00000000		
0EC: 00000000		
0F0: 00000000		
0F4: 00000000		
0F8: 00000000		
0FC: 00000000		
000: 77FF0094		
004: C3ED0004		
008: 679DFFFC		
00C: C3ED0004		
010: 677DFFFC		
014: 837DF800		
018: C3ED0004		
01C: 641DFFFC		
020: C3ED0004		
024: 643DFFFC		
028: C3ED0004		
02C: 645DFFFC		
030: C3ED0004		

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*

intitled
untitled
beta.uasm
intio.uasm

### REGISTERS

R0:00000000	R8:00000000	R16:00000000	R24:00000000
R1:00000000	R9:00000000	R17:00000000	R25:00000000
R2:00000000	R10:00000000	R18:00000000	R26:00000000
R3:00000000	R11:00000000	R19:00000000	BP:00000000
R4:00000000	R12:00000000	R20:00000000	LP:80000268
R5:00000000	R13:00000000	R21:00000000	SP:0000029C
R6:00000000	R14:00000000	R22:00000000	XP:00000000
R7:00000000	R15:00000000	R23:00000000	R31:00000000

### INSTRUCTIONS (SUPERVISOR MODE)

248:639DFFFC	LD(SP,-4,LP)
24C:C3BDFFFC	ADDC(SP,-4,SP)
250:6FFC0000	JMP(LP)
254:C3BF029C	intiofin: ADDC(R31,658,SP)
258:77FF0002	BR(debut)
25C:00000000	b: HALT()
260:00000000	a: HALT()
264:779F0001	debut: BR(main,LP)
268:00000000	HALT()
26C:C3BD0004	main: ADDC(SP,4,SP)
270:679DFFFC	ST(LP,-4,SP)
274:C3BD0004	ADDC(SP,4,SP)
278:677DFFFC	ST(BP,-4,SP)
27C:837DF800	ADD(SP,R31,BP)
280:C3BD0000	ADDC(SP,0,SP)
284:C7BD0000	SUBC(SP,0,SP)

### STACK

238:601DFFFC
23C:C3BDFFFC
240:637DFFFC
244:C3BDFFFC
248:639DFFFC
24C:C3BDFFFC
250:6FFC0000
254:C3BF029C
258:77FF0002
25C:00000000
260:00000000
264:779F0001
268:00000000
26C:C3BD0004
270:679DFFFC
274:C3BD0004
278:677DFFFC
27C:837DF800
280:C3BD0000
284:C7BD0000
288:637DFFFC
28C:C3BDFFFC
290:639DFFFC
294:C3BDFFFC
298:6FFC0000
29C:80000268

### MEM[0x2a0]

26C:C3BD0004
270:679DFFFC
274:C3BD0004
278:677DFFFC
27C:837DF800
280:C3BD0000
284:C7BD0000
288:637DFFFC
28C:C3BDFFFC
290:639DFFFC
294:C3BDFFFC
298:6FFC0000
29C:80000268
2A0:00000000
2A4:00000000
2A8:00000000
2AC:00000000
2B0:00000000
2B4:00000000
2B8:00000000
2BC:00000000
2C0:00000000
2C4:00000000
2C8:00000000
2CC:00000000
2D0:00000000

## TDS g  n  r  

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

## Arbre g  n  r  

```
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); //écriture
}

```

#### *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:

# Running simulation... press stop to view state

? 5

## REGISTERS

R0:0000000F	R8:00000000	R16:00000000	R24:00000000
R1:00000005	R9:00000000	R17:00000000	R25:00000000
R2:0000000A	R10:00000000	R18:00000000	R26:00000000
R3:0000000F	R11:00000000	R19:00000000	BP:00000000
R4:00000000	R12:00000000	R20:00000000	LP:80000264
R5:00000000	R13:00000000	R21:00000000	SP:0000030C
R6:00000000	R14:00000000	R22:00000000	XP:00000000
R7:00000000	R15:00000000	R23:00000000	R31:00000000

## INSTRUCTIONS (SUPERVISOR MODE)

244:C3BDFFFC	ADDC(SP,-4,SP)
248:639DFFFC	LD(SP,-4,LP)
24C:C3BDFFFC	ADDC(SP,-4,SP)
250:6FFC0000	JMP(LP)
254:C3BF030C	intiofin: ADDC(R31,780,SP)
258:77FF0001	BR(debut)
25C:00000005	b: CLICK()
260:779F0001	debut: BR(main,LP)
264:00000000	HALT()
268:C3BD0004	main: ADDC(SP,4,SP)
26C:679DFFFC	ST(LP,-4,SP)
270:C3BD0004	ADDC(SP,4,SP)
274:677DFFFC	ST(BP,-4,SP)
278:837DF800	ADD(SP,R31,BP)
27C:C3BD0000	ADDC(SP,0,SP)
280:C3BD0004	ADDC(SP,4,SP)

PC →

## STACK

2A8:C3BD0004
2AC:641DFFFC
2B0:C01F000A
2B4:C3BD0004
2B8:641DFFFC
2BC:605DFFFC
2C0:C3BDFFFC
2C4:603DFFFC
2C8:C3BDFFFC
2CC:80611000
2D0:C3BD0004
2D4:647DFFFC
2D8:601DFFFC
2DC:C3BDFFFC
2E0:C3BD0004
2E4:641DFFFC
2E8:779FFF9A
2EC:601DFFFC
2F0:C3BDFFFC
2F4:C7BD0000
2F8:637DFFFC
2FC:C3BDFFFC
300:639DFFFC
304:C3BDFFFC
308:6FFC0000
30C:80000264

SP →

## MEM[0x33c]

308:6FFC0000
30C:80000264
310:00000000
314:0000000F
318:800002EC
31C:00000314
320:0000000F
324:00000005
328:0000000A
32C:0000000F
330:00000000
334:00000000
338:00000005
33C:00000001
340:00000000
344:00000000
348:00000000
34C:00000000
350:00000000
354:00000000
358:00000000
35C:00000000
360:00000000
364:00000000
368:00000000
36C:00000000

? 5

15

## 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  
  
CHOVE(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)  
    CHOVE(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)
```

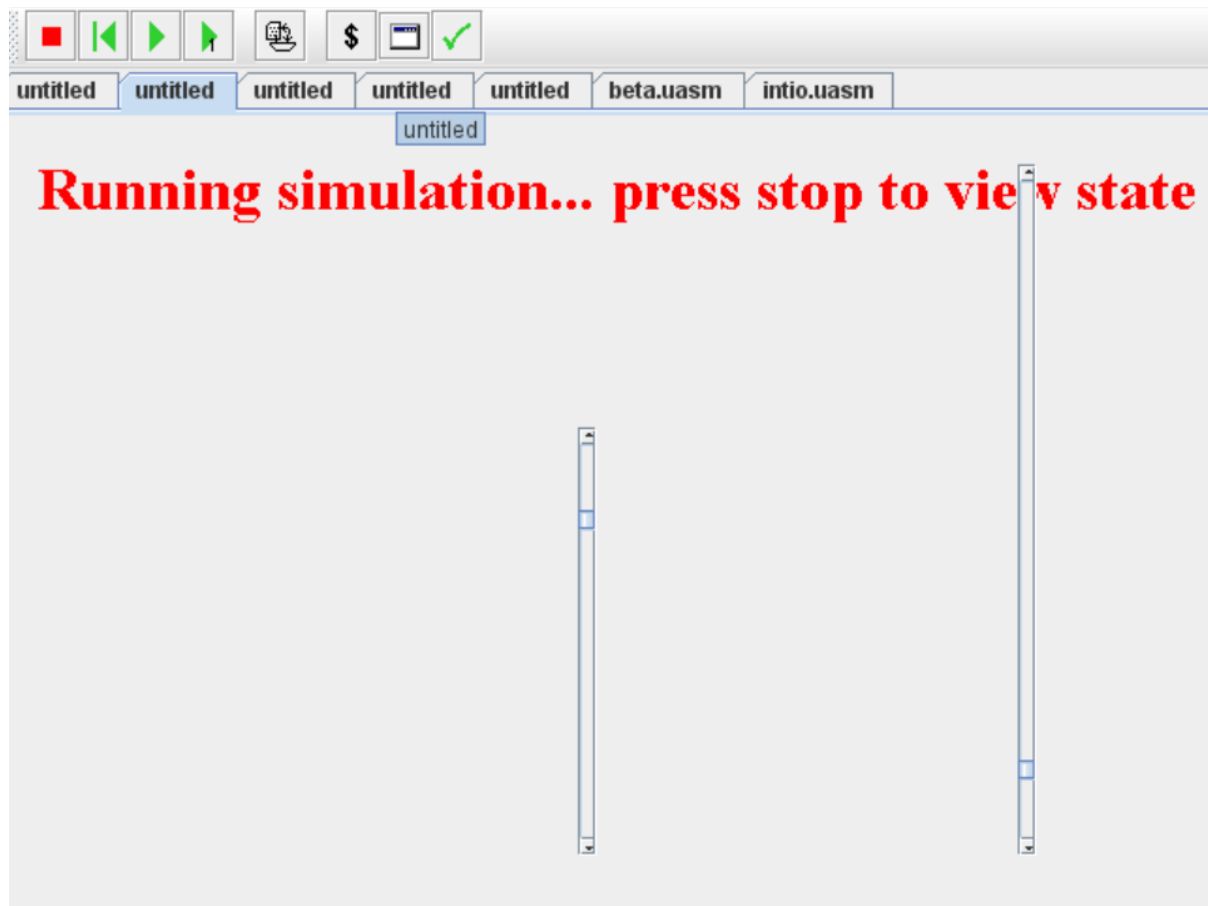
```
    POP(BP)
```

```
    POP(LP)
```

```
    RTN()
```

```
pile:
```

## Exécution dans Bsim



Registers	Instructions (SUPERVISOR MODE)	Stack	MEM[0x398]
R0: 00000005 R8: 00000000 R16: 00000000 R24: 00000000	248: 639DFFFC LD(SP, -4, LP)	308: 641DFFFC	364: C3BDFFFC
R1: 00000000 R9: 00000000 R17: 00000000 R25: 00000000	24C: C3BDFFFC ADDC(SP, -4, SP)	30C: 601DFFFC	368: 6FFC0000
R2: 00000005 R10: 00000000 R18: 00000000 R26: 00000000	250: 6FFC0000 JMP(LP)	310: C3BDFFFC	36C: 80000268
R3: 00000000 R11: 00000000 R19: 00000000 R27: 00000000	254: C3BF036C intiofin: ADDC(R31, 876, SP)	314: C3BD0004	370: 00000000
R4: 00000000 R12: 00000000 R20: 00000000 R28: 80000268	258: 77FF0002 BR(debut)	318: 641DFFFC	374: 00000005
R5: 00000000 R13: 00000000 R21: 00000000 R29: 0000036C	25C: 00000005 b: CLICK()	31C: 779FFF8D	378: 8000034C
R6: 00000000 R14: 00000000 R22: 00000000 R30: 00000000	260: 00000004 a: TIME()	320: 601DFFFC	37C: 00000374
R7: 00000000 R15: 00000000 R23: 00000000 R31: 00000000	264: 779F0001 debut: BR(main, LP)	324: C3BDFFFC	380: 00000005
	PC → 268: 00000000 HALT()	328: 77FF000A	384: 00000000
	26C: C3BD0004 main: ADDC(SP, 4, SP)	32C: 601F025C	388: 00000005
	270: 679DFFFC ST(LP, -4, SP)	330: C3BD0004	38C: 00000000
	274: C3BD0004 ADDC(SP, 4, SP)	334: 641DFFFC	390: 00000000
	278: 677DFFFC ST(BP, -4, SP)	338: 601DFFFC	394: 00000000
	27C: 837DF800 ADD(SP, R31, BP)	33C: C3BDFFFC	398: 00000005
	280: C3BD0000 ADDC(SP, 0, SP)	340: C3BD0004	39C: 00000000
	284: C3BD0004 ADDC(SP, 4, SP)	344: 641DFFFC	3A0: 00000000
		348: 779FFF82	3A4: 00000000
		34C: 601DFFFC	3A8: 00000000
		350: C3BDFFFC	3AC: 00000000
		354: C7BD0000	3B0: 00000000
		358: 637DFFFC	3B4: 00000000
		35C: C3BDFFFC	3B8: 00000000
		360: 639DFFFC	3BC: 00000000
		364: C3BDFFFC	3C0: 00000000
		368: 6FFC0000	3C4: 00000000
		36C: 80000268	3C8: 00000000

SP → 36C: 80000268

## 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:



untitled

untitled

untitled

untitled

untitled

beta.uasm

intio.uasm

REGISTERS

R0: 000002D0	R8: 00000000	R16: 00000000	R24: 00000000
R1: 00000000	R9: 00000000	R17: 00000000	R25: 00000000
R2: 00000001	R10: 00000000	R18: 00000000	R26: 00000000
R3: 00000000	R11: 00000000	R19: 00000000	BP: 00000000
R4: 00000000	R12: 00000000	R20: 00000000	LP: 80000268
R5: 00000000	R13: 00000000	R21: 00000000	SP: 000003E0
R6: 00000000	R14: 00000000	R22: 00000000	XP: 00000000
R7: 00000000	R15: 00000000	R23: 00000000	R31: 00000000

INSTRUCTIONS (SUPERVISOR MODE)

248: 639DFFFC	LD(SP, -4, LP)
24C: C3BDFFFC	ADDC(SP, -4, SP)
250: 6FFC0000	JMP(LP)
254: C3BF03E0	intiofin: ADDC(R31, 944, SP)
258: 77FF0002	BR(debut)
25C: 000002D0	res: illop
260: 00000001	a: RDCHAR()
264: 779F0001	debut: BR(main, LP)
PC → 268: 00000000	HALT()
26C: C3BD0004	main: ADDC(SP, 4, SP)
270: 679DFFFC	ST(LP, -4, SP)
274: C3BD0004	ADDC(SP, 4, SP)
278: 677DFFFC	ST(BP, -4, SP)
27C: 837DF800	ADD(SP, R31, BP)
280: C3BD0000	ADDC(SP, 0, SP)
284: C3BD0004	ADDC(SP, 4, SP)

STACK

34C: 603DFFFC
350: C3BDFFFC
354: 94620800
358: C3BD0004
35C: 647DFFFC
360: 603DFFFC
364: C3BDFFFC
368: 77E10001
36C: 77FFFFCF
370: 601F025C
374: C3BD0004
378: 641DFFFC
37C: 601DFFFC
380: C3BDFFFC
384: C3BD0004
388: 641DFFFC
38C: 779FFF71
390: 601DFFFC
394: C3BDFFFC
398: C7BD0000
39C: 637DFFFC
3A0: C3BDFFFC
3A4: 639DFFFC
3A8: C3BDFFFC
3AC: 6FFC0000
SP → 3B0: 80000268

MEM(0x3e4)

3B0: 80000268
3B4: 00000000
3B8: 000002D0
3BC: 80000390
3C0: 000003E8
3C4: 000002D0
3C8: 00000000
3CC: 00000001
3D0: 00000000
3D4: 00000000
3D8: 00000000
3DC: 00000000
3E0: 00000002
3E4: 00000007
3E8: 00000000
3EC: 00000000
3F0: 00000000
3F4: 00000000
3F8: 00000000
3FC: 00000000

? 6

720

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

untitled	untitled	untitled	untitled	untitled	untitled	beta.uasm	intio.uasm
----------	----------	----------	----------	----------	----------	-----------	------------

**REGISTERS**

R0: 0000000F	R8: 00000000	R16: 00000000	R24: 00000000
R1: 00000005	R9: 00000000	R17: 00000000	R25: 00000000
R2: 0000000A	R10: 00000000	R18: 00000000	R26: 00000000
R3: 0000000F	R11: 00000000	R19: 00000000	BP: 00000000
R4: 00000000	R12: 00000000	R20: 00000000	LP: 80000260
R5: 00000000	R13: 00000000	R21: 00000000	SP: 00000374
R6: 00000000	R14: 00000000	R22: 00000000	XP: 00000000
R7: 00000000	R15: 00000000	R23: 00000000	R31: 00000000

**INSTRUCTIONS (SUPERVISOR MODE)**

240: 637DFFFC	LD(SP, -4, BP)
244: C3BDFFFC	ADDC(SP, -4, SP)
248: 639DFFFC	LD(SP, -4, LP)
24C: C3BDFFFC	ADDC(SP, -4, SP)
250: 6FFC0000	JMP(LP)
254: C3BF0374	intiofin: ADDC(R31, 884, SP)
258: 77FF0000	BR(debut)
25C: 779F0023	debut: BR(main, LP)
260: 00000000	HALT()
264: C3BD0004	addition: ADDC(SP, 4, SP)
268: 679DFFFC	ST(LP, -4, SP)
26C: C3BD0004	ADDC(SP, 4, SP)
270: 677DFFFC	ST(BP, -4, SP)
274: 837DF800	ADD(SP, R31, BP)
278: C3BD0004	ADDC(SP, 4, SP)
27C: 601BFFFC	LD(BP, -16, R0)

**STACK**

310: 641DFFFC
314: C01F000A
318: C3BD0004
31C: 641DFFFC
320: 779FFFD0
324: C7BD0008
328: 601DFFFC
32C: C3BDFFFC
330: 641B0000
334: 601B0000
338: C3BD0004
33C: 641DFFFC
340: 601DFFFC
344: C3BDFFFC
348: C3BD0004
34C: 641DFFFC
350: 779FFF80
354: 601DFFFC
358: C3BDFFFC
35C: C7BD0004
360: 637DFFFC
364: C3BDFFFC
368: 639DFFFC
36C: C3BDFFFC
370: 6FFC0000
374: 80000260

**MEM[0x328]**

374: 80000260
378: 00000000
37C: 0000000F
380: 0000000F
384: 80000354
388: 0000037C
38C: 0000000F
390: 00000005
394: 0000000A
398: 0000000F
39C: 00000000
3A0: 00000000
3A4: 00000005
3A8: 00000001
3AC: 00000000
3B0: 00000000
3B4: 00000000
3B8: 00000000
3BC: 00000000
3C0: 00000000
3C4: 00000000
3C8: 00000000
3CC: 00000000
3D0: 00000000
3D4: 00000000
3D8: 00000000

15

## 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/$
│       └─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 l_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 paramÃtres dÃclarÃs dans une fonction

```

```

l_param_declare ::= l_param_declare:lp VIRGULE param_declare:p {::;}
    | param_declare:p {::;}
    | {::;}
;

// Un paramètre déclaré dans une fonction
param_declare ::= ENTIER IDF:idf {::;}
;

// Liste de déclaration locale
l_declaration_locale ::= l_declaration_locale declaration_locale
    | declaration_locale {::;}
;

// déclaration locale
declaration_locale ::= ENTIER IDF PVIRGULE {::;}
    | ENTIER IDF AFFE NUM PVIRGULE {::;}
;

// liste de déclaration globale
l_declaration_globale ::= l_declaration_globale declaration_globale: {::;}
    | declaration_globale {::;}
;

// Déclaration globale avant la création 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 paramÃtres )
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,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: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 allons 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.