République Algérienne Démocratique et Populaire

الجمهورية الجزائرية الديمقراطية الشعبية

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

وزارة التعليم العالي والبحث العلمي

المدرسة الوطنية العليا للاعلام الآلي

المعهد العالي للتكوين في الاعلام الآلي مسبقا

Ecole nationale Supérieure d'Informatique

ex. INI (Institut National de formation en Informatique)

Mini-Projet : COMPIL

2ème année Cycle Supérieur (2CS)

Option : Systèmes Informatiques (SQ)

Réalisation d'un traducteur pour le langage Mini-C

Réalisé par :

Borhaneddine HAMADOU (SIQ2)

Mohamed Zakaria MILOUDI (SIQ2)

Sous la supervision de :

Dr. Chebieb A.

Promotion : 2021 - 2022

1 Introduction

Le But de ce Mini-Projet est de développer un traducteur pour le langage Mini-C. Le compilateur doit produire un code intermédiaire sous forme de quadruplets correspondant au code source donné et pour cette dernière nous avons choisir le code assembleur comme code intermédiaire. Le présent document compris les différents tokens ou unités lexicaux, la grammaire LL(1) utilisé ainsi que le schéma de traduction.

2 Les Tokens

```
TOKEN : {
    <TYPE: <INT> | <FLOAT>  >          |
    <IF:   "if"  >                     |
    <ELSE:   "else"  >                 |
    <#INT: "int" >                     |
    <#FLOAT: "float" >                 |
    <WHILE: "while" >                  |
    <FOR: "for" >
}

TOKEN: {
    <ID: <LETTER> (<LETTER> | <DIGIT> | <ZERO>)* >                |
    <NUMBER: <DIGIT> (<DIGIT> | <ZERO>)* | <ZERO> >               |
    <#LETTER: ["a"-"z","A"-"Z"] >                                 |
    <#DIGIT: ["1"-"9"] >                                          |
    <#ZERO: "0" >
}

TOKEN: {
    <GPAR: "(" >        |
    <DPAR: ")" >        |
    <VIRGULE:     "," > |
    <GACC: "{" >        |
    <DACC: "}" >        |
    <POINTVIRGULE: ";" > |
    <AFFECT:    "=" > |
    <EQ:       "==" > |
    <LT:       "<"  > |
    <LTE:      "<=" > |
    <GT:       ">"  > |
    <GTE:      ">=" > |
    <NEQ:      "!=" > |
    <ADD:       "+" > |
    <SUB:       "-" > |
    <MUL:       "*" > |
    <DIV:       "/" >
}
```

3 La grammaire sous forme BNF

```
Start ::= (<Function>)* #

Function ::= <Type> identifier "(" <ArgList> ")" <StatementBlockDef>

ArgList ::= <Arg> ("," <Arg>)*

Arg ::= <Type> identifier

varDefineDef ::= <Type> identifier ( "=" <ExprAssign> )? ( "," <varList> )*

varList ::= identifier ( "=" <ExprAssign> )?

StatementBlockDef ::= "{" (<Stmt>)* "}"

Stmt ::= <varDefineDef> ";" | <Expr> ";" | <IfStmt> | <ForStmt> | <WhileStmt>

IfStmt ::= "if" "(" <ExprAssign> <Rvalue> ")" <StatementBlockDef> ( "else"
           <StatementBlockDef> | <Stmt> )

Expr ::= <ExprAssign> ( <Affec> )?

Affec ::= "=" <ExprAssign> | <Rvalue>

ExprAssign ::= ( identifier | number ) ( ( "+" | "-" | "*" | "/" )
               <ExprAssign> )? | "(" <ExprAssign> ")"

Rvalue ::= ( "<" | ">" | "==" | "!=" | " >=" | "<=" ) <ExprAssign>

WhileStmt ::= "while" "(" <ExprAssign> <Rvalue> ")" ( <StatementBlockDef>
              | <Stmt> )
```

```
ForStmt ::= "for" "(" ( <varDefineDef> )? ";" ( <ExprAssign> <Rvalue> )? ";"
        ( ( identifier "=" <ExprAssign> ) )? ")" <StatementBlockDef>
```

4 Le Schéma de traduction : Les Routines Sémantiques

4.1 La Boucle For

```
ForStmt ::= "for" "(" ( <varDefineDef> )? ";" M1 ( <ExprAssign> <Rvalue> )? M2";"
            M3( ( identifier M4"=" <ExprAssign> M5) )? M6")" M7
            <StatementBlockDef> M8

M1 : GenQuad( "ETQ" + currentFun + quadC + ":" , "", ""); quadC++;

M2 : GenQuad( "save" , "", "" ); quadC++;

M3 : GenQuad( "ETQ" + currentFun + quadC + ":" , ""); quadC++;

M4 : SAUVEGARDER LA VARIABLE

M5 : GenQuad( "mov" , "AX", ADDR); quadC++;
     GenQuad( "mov" , ADDR, "AX"); quadC++;

M6 : GenQuad( "save" , "", ""); quadC++;

M7 : GenQuad( "ETQ" + currentFun + quadC + ":" , "", ""); quadC++;

M8 : codeGen( "jmp" , "ETQ" + I, ""); quadC++;
     codeGen( "jmp" , "ETQ" + J, ""); quadC++;
     GenQuad( "ETQ" + currentFun + quadC + ":" , "");
     quadC++;
```

4.2 La Boucle While

```
WhileStmt ::= "while" "(" M1  <ExprAssign> <Rvalue> ")" ( <StatementBlockDef>
            | <Stmt> M2)

M1 : GenQuad( "ETQ" + currentFun + quadC + ":" , "", ""); quadC++;
M2 : GenQuad( "ETQ" + currentFun + quadC + ":" , "");
     quadC++;
     GenQuad( "ETQ" + currentFun + quadC + ":" , "",
     ""); quadC++;
```

4.3 La Structure Conditionnelle

```
IfStmt ::= "if" "(" <ExprAssign> <Rvalue> ")" <StatementBlockDef> ( "else"
            M1 <StatementBlockDef> | <Stmt> M2)|M3

M1 : GenQuad( "save" , "", "" ); quadC++;

M2 : GenQuad( "jmp" , "ETQ" + I, ""); quadC++;

M3 : GenQuad( "ETQ" + currentFun + quadC + ":" , "",
     ""); quadC++;
```

4.4 Les Expressions Arithmétiques

```
ExprAssign ::= ( identifier M1| number M2 ) ( ( "+"M3 | "-" M4| "*" M5| "/" M6)
               <ExprAssign> )?| "("  <ExprAssign> ")"

M1 : GenQuad( "mov" , AX, VAL); quadC++;
     GenQuad( "mov" , ADDR, AX); quadC++;

M2 : GenQuad( "mov" , AX, VAL); quadC++;
     GenQuad( "mov" , ADDR, AX); quadC++;

M3 : GenQuad( "mov" , CX, ADDR1); quadC++;
     GenQuad( "add" , CX, ADDR2); quadC++;
     GenQuad( "mov" , ADDR1, CX); quadC++;

M4 : GenQuad( "mov" , CX, ADDR1); quadC++;
     GenQuad( "sub" , CX, ADDR2); quadC++;
     GenQuad( "mov" , ADDR1, CX); quadC++;

M5 : GenQuad( "mov" , CX, ADDR1); quadC++;
     GenQuad( "mov" , AL, ADDR2); quadC++;
     GenQuad( "mul" , CL, ""); quadC++;
     GenQuad( "mov" , ADDR1, AX); quadC++;

M6 : GenQuad( "mov" , CX, ADDR1); quadC++;
     GenQuad( "mov" , AL, ADDR2); quadC++;
     GenQuad( "div" , CL, ""); quadC++;
     GenQuad( "mov" , ADDR1, AX); quadC++;
```

4.5 La Comparaison

```
Rvalue ::= ( op = "<"M1 | ">"M2 | "=="M3 | "!="M4 | ">="M5 | "<="M6 )
             <ExprAssign>

M1 : GenQuad( "mov" , AX, ADDR ); quadC++;
     GenQuad( "mov" , BX, ADDR ); quadC++;
     GenQuad( "cmp" , AX, BX ); quadC++;
     GenQuad( "jge" , "ETQ"+quadC, "" ); quadC++;

M2 : GenQuad( "mov" , AX, ADDR ); quadC++;
     GenQuad( "mov" , BX, ADDR ); quadC++;
     GenQuad( "cmp" , AX, BX ); quadC++;
     GenQuad( "jle" , "ETQ"+quadC, "" ); quadC++;

M3 : GenQuad( "mov" , AX, ADDR ); quadC++;
     GenQuad( "mov" , BX, ADDR ); quadC++;
     GenQuad( "cmp" , AX, BX ); quadC++;
     GenQuad( "jne" , "ETQ"+quadC, "" ); quadC++;

M4 : GenQuad( "mov" , AX, ADDR ); quadC++;
     GenQuad( "mov" , BX, ADDR ); quadC++;
     GenQuad( "cmp" , AX, BX ); quadC++;
     GenQuad( "je" , "ETQ"+quadC, "" ); quadC++;

M5 : GenQuad( "mov" , AX, ADDR ); quadC++;
     GenQuad( "mov" , BX, ADDR ); quadC++;
     GenQuad( "cmp" , AX, BX ); quadC++;
     GenQuad( "jl" , "ETQ"+quadC, "" ); quadC++;

M6 : GenQuad( "mov" , AX, ADDR ); quadC++;
     GenQuad( "mov" , BX, ADDR ); quadC++;
     GenQuad( "cmp" , AX, BX ); quadC++;
     GenQuad( "jg" , "ETQ"+quadC, "" ); quadC++;
```

4.6 L'affectation

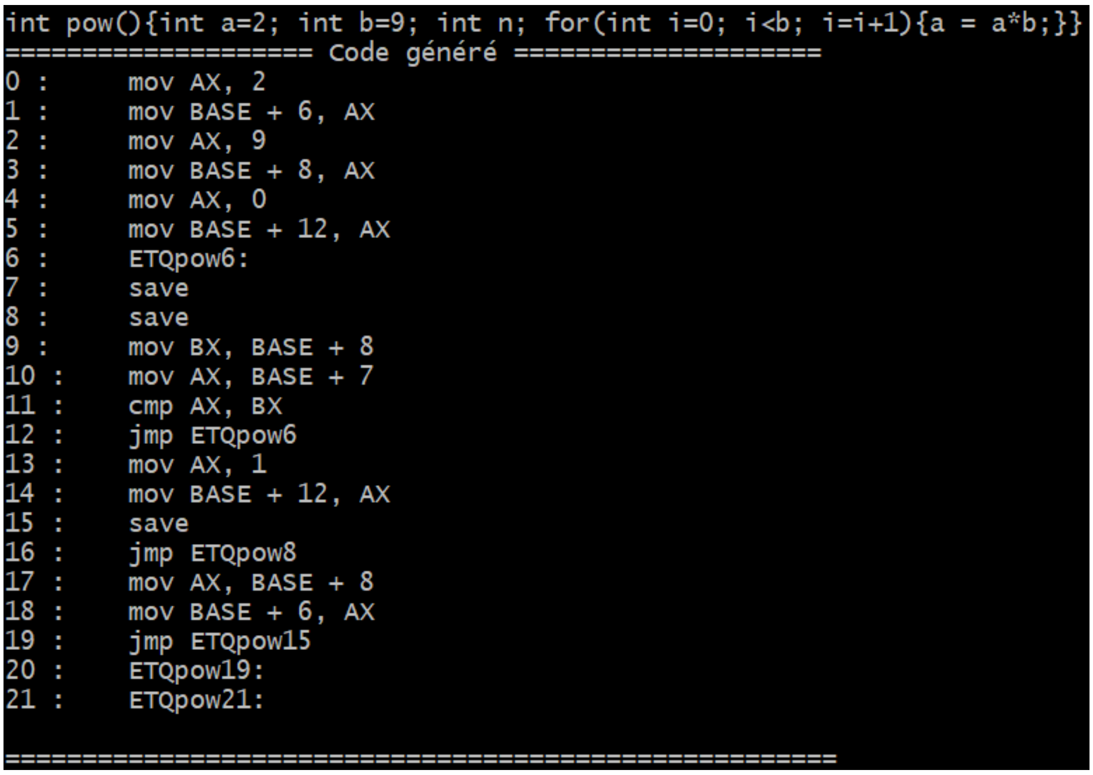
```
Affec ::= "=" <ExprAssign> M1 | <Rvalue>


M1 : GenQuad( "mov" , AX, VAL); quadC++;
     GenQuad( "mov" , ADDR, AX); quadC++;
```

5 Tests & Résultats

5.1 La Fonction Puissance

```
int pow(){int a=2; int b=9; int n; for(int i=0; i<b; i=i+1){a = a*b;}}
```



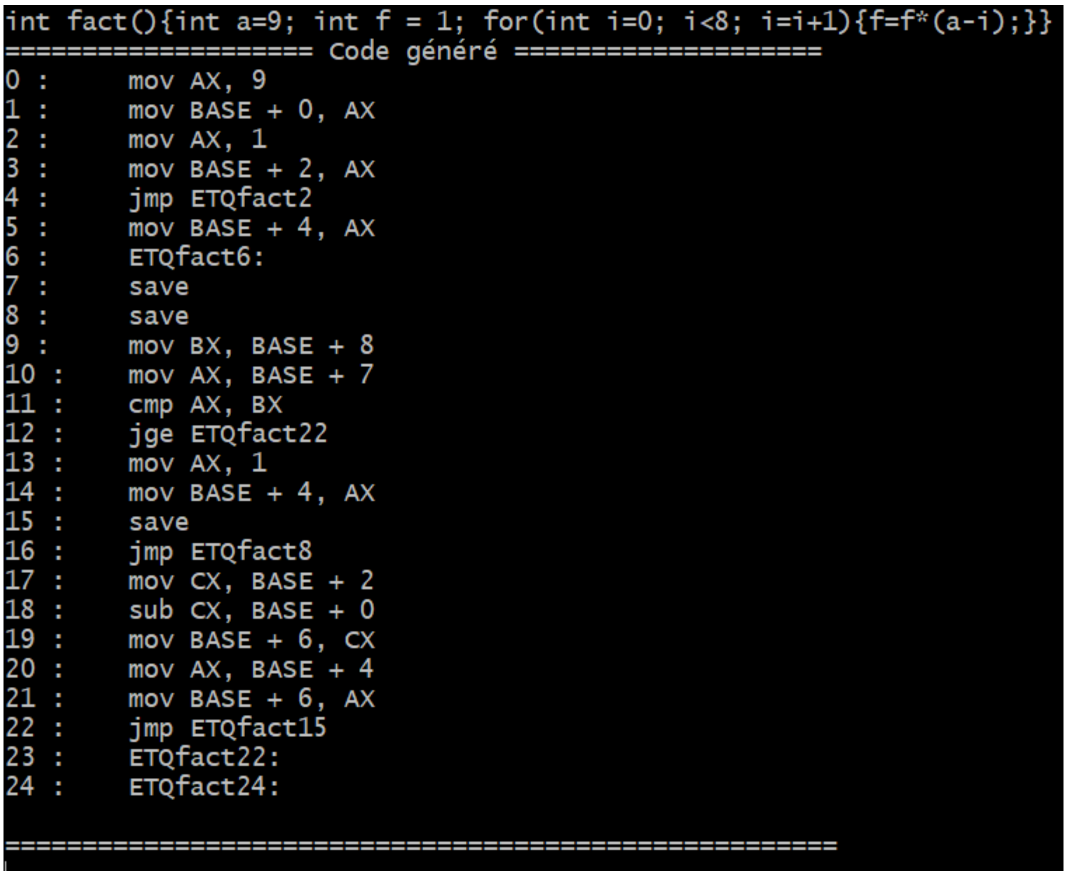
```
int pow(){int a=2; int b=9; int n; for(int i=0; i<b; i=i+1){a = a*b;}}
==================== Code généré ====================
0 :      mov AX, 2
1 :      mov BASE + 6, AX
2 :      mov AX, 9
3 :      mov BASE + 8, AX
4 :      mov AX, 0
5 :      mov BASE + 12, AX
6 :      ETQpow6:
7 :      save
8 :      save
9 :      mov BX, BASE + 8
10 :     mov AX, BASE + 7
11 :     cmp AX, BX
12 :     jmp ETQpow6
13 :     mov AX, 1
14 :     mov BASE + 12, AX
15 :     save
16 :     jmp ETQpow8
17 :     mov AX, BASE + 8
18 :     mov BASE + 6, AX
19 :     jmp ETQpow15
20 :     ETQpow19:
21 :     ETQpow21:

======================================================
```

Figure 1: La Fonction Puissance

5.2 La Fonction Factorielle

```
int fact(){int a=9; int f = 1; for(int i=0; i<8; i=i+1){f=f*(a-i);}}
```



```
int fact(){int a=9; int f = 1; for(int i=0; i<8; i=i+1){f=f*(a-i);}}
==================== Code généré ====================
0 :     mov AX, 9
1 :     mov BASE + 0, AX
2 :     mov AX, 1
3 :     mov BASE + 2, AX
4 :     jmp ETQfact2
5 :     mov BASE + 4, AX
6 :     ETQfact6:
7 :     save
8 :     save
9 :     mov BX, BASE + 8
10 :    mov AX, BASE + 7
11 :    cmp AX, BX
12 :    jge ETQfact22
13 :    mov AX, 1
14 :    mov BASE + 4, AX
15 :    save
16 :    jmp ETQfact8
17 :    mov CX, BASE + 2
18 :    sub CX, BASE + 0
19 :    mov BASE + 6, CX
20 :    mov AX, BASE + 4
21 :    mov BASE + 6, AX
22 :    jmp ETQfact15
23 :    ETQfact22:
24 :    ETQfact24:

======================================================
```

Figure 2: La Fonction Factorielle