U.S.T.H.B
Faculté Informatique
Module de Compilation2
1ère Année Master IL et MIV **Année Universitaire : 2021/2022**

EPREUVE DE COMPILATION

Exercice 1:(6points)

1. Donner les différentes étapes du processus de compilation, en mentionnant clairement les objectifs de chacune.

2. Qu'entend nous par compilateur mono passe et compilateur multi passes ?

3. Quel est le rôle d'une déclaration de tableau dans un langage de programmation ?

4. Donner clairement la différence entre un langage à allocation statique et allocation dynamique.

5. Nous supposons une portion de programme dans laquelle nous déclarons un tableau dont les bornes sont variables. Détaillez ce qui se passe dans la phase allocation et la phase exécution.

6. La phase optimisation de code est-elle nécessaire et dans quelles étapes de la compilation, intervient-elle ?

7. De quoi a-t-on besoin pour faire une bonne génération de code objet.

8. Donner le rôle de la fonction Getinacc.

Exercice 2:(6points)

Soit l'instruction :

Execute (B1 : S1; B2 : S2; …. ;Bn : Sn).

Fonctionnement de l'instruction :

Si B1 est vrai alors Exécuter S1 ;
Si B2 est vrai alors Exécuter S2 ;
……

Si Bn est vrai alors Exécuter Sn ;

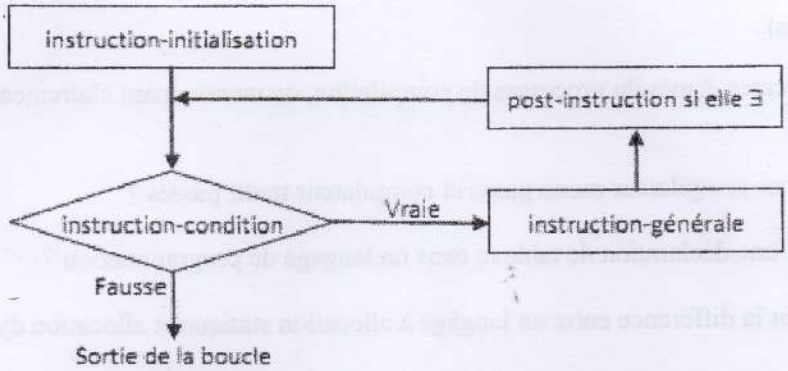
a. Donner la grammaire syntaxique de l'instruction.
b. Donner le schéma de traduction de l'instruction en utilisant les quadruplets avec une analyse ascendante, sachant que n>0.

Exercice 3 : (8points)

Considérer l'instruction **FOR** apparentée à l'instruction **FOR** du langage C et dont les productions de génération sont données ci-après :

<instruction-FOR> →**FOR** '(' <instruction-initialisation> ' ; ' <instruction-condition> ' ; ' ')' <instruction-générale>
| **FOR** '(' <instruction-initialisation> ' ; ' <instruction-condition> ' ; ' <post-instruction> ')' <instruction-générale>

L'organigramme de cette instruction est :



Exemple :

```
FOR ( instr1;cond1;instr2)
{ instr3;
  FOR (instr4;cond2;)
Instr5;}
```

- **L'instruction-générale** permet d'affecter à une variable une constante, une variable où bien le résultat d'une expression arithmétique (+, -, x , /).
- **L'instruction-condition** permet de comparer une variable avec une variable, où bien une variable avec une constante (exemple : x>12, Y>X).
- Une constante entière est une suite de chiffres. Elle peut être signée ou non signée tel que sa valeur est entre -32768 et 32767. Si la constante entière est signée, elle doit être mise entre parenthèses.
- Un identificateur est une suite alpha numérique qui commence par une lettre majuscule suivie d'une suite de chiffres et lettres minuscules.Un identificateur ne doit pas contenir plus de 8 caractères.

1. Donner les expressions régulières nécessaires écrites en langage Flex.
2. Donner les règles de traduction nécessaires écrites en langage Flex.
   a. A quoi sert la table des symboles dans l'analyse syntaxico-sémantique?
   b. Donner la grammaire de l'expression ci-dessus, ainsi que le programme permettant de générer les quadruplets de **l'instruction FOR** dans le cas d'une analyse ascendante. Prendre en considération les imbrications d'instructions FOR. En utilisant le langage Bison.
   d. Donnez le programme permettant de générer le code machine correspondant au code intermédiaire précédant sachant qu'on utilise la machine Assembleur 8086 composée de 4 registres vue en TP.

Corrigé END Compil 2021/2022
IL et IV

Exo1

(1) Tous les étapes du processus de compilation, en expliquant chaque étape. (1pt)

(2) Un compilateur monopasse, est un compilateur où toutes les étapes se font en une seule fois. Multipasse : plusieurs étapes. (0,5pt)

(3) Déclaration d'un tableau, c'est donner le type des éléments du tableau, qui nous permet de prévoir l'espace mémoire nécessaire pour le tableau. (réservation en compilation dans le cas des langages statiques et dynamique ~~dans~~ à exécution). (0,5pt)

(4) Langage à allocation statique, l'espace mémoire nécessaire aux données, est réservé déjà en compilation les adresses relatives sont connues en compilation et la taille de chaque ~~le~~ pendant l'exécution, les adresses des objets sont connues par rapport à un début de zone de données. Langage à allocation dynamique : l'espace nécessaire à l'exécution des programmes est réservé de façon dynamique à chaque exécution. Une fois, l'exécution achevée, ~~les~~ données n'existent plus. (01pt)

(5) Déclaration d'un tableau à taille variable, on ne peut faire une réservation d'espace pour les ~~programmes~~ (0,5pt)

~~Les~~ données pendant la compilation - Cela se fait pendant la phase exécution - Ou bien pour ce R et pendant l'exécution, on le met à jour

5) — La phase optimisation de code est optionnelle dans les compilateurs - Lorsqu'elle est présente, elle peut aller à tous les niveaux de l'analyse lexicale jusqu'à la phase génération de code objet (0,1 pt)

(7) Pour faire une bonne génération de code objet, nous avons besoin d'une machine dotée d'un bon et riche jeu d'instructions et assez de registres et ACC (0,5 pt)

(8) La fonction Getinacc permet de connaître l'état de l'accumulateur, lors de la génération du code objet, à travers la variable acc, qui est une variable de la compilation - (0,1 pt)

Exercice 2 :

Execute ($B_1 : S_1$; ..... , $B_n : S_n$)

a- Grammaire syntaxique de l'instruction : (1 point)

<Inst - Execute> → Execute (<list-cond-inst>)

<list-cond-inst> → <list-cond-inst> ; <cond> : <inst> / <cond> : <inst>

b- ① Code Intermédiaire sous forme de quadruplés :

quadruplés de B_1

(BZ, dés-Bz, <B1>.temp,)

quadruplés de S_1

dés-Bz → quadruplés de B_2

(1,2 points)

(BZ, dés-B3, <B2>.temp,)

quadruplés de S_2

.

.

.

dés-Bn → quadruplés de B_n

(BZ, Fin, <Bn>.temp,)

quadruplés de S_n

Fin →

(2) Grammaire associée :

<Inst. Execute> → Execute(<Lst. cmd-nst>)

<Lst. cmd-nst> → A <inst> (R2) B <inst> (R2)

A → <Lst. cmd-nst>; <cmd> : (R1)

B → <cmd> : (R1)

1,25 points

(3) les routines sémantiques :

Routine (R1)

début

QUAD(qc) := (BZ, , <cmd>.fsg,);

SauV-Bz := qc ;

qc := qc+1;

Fin.

1,25 points

Routine (R2)

début

QUAD(SauV-Bz, 2) := qc ;

Fin

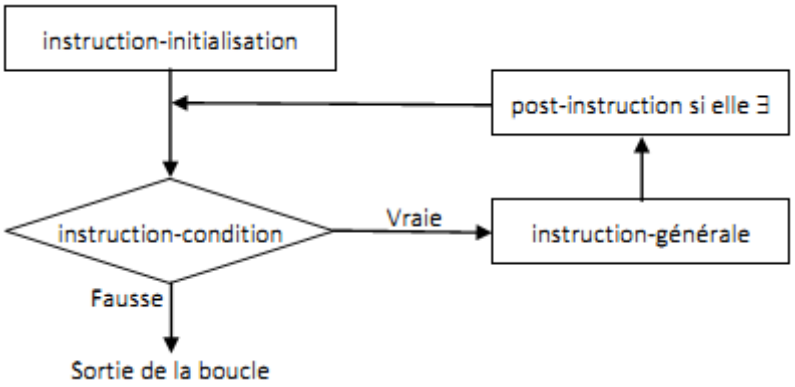
1,25 points

Exercice 3 : (8points)

Considérer l'instruction **FOR** apparentée à l'instruction **FOR** du langage C et dont les productions de génération sont données ci-après :

<**instruction-FOR**> **FOR** '(' <instruction-initialisation> ';' <instruction-condition> ';' ')' <instruction-générale> | **FOR** '(' <instruction-initialisation> ';' <instruction-condition> ' ;' <post-instruction> ')' <instruction-générale>

L'organigramme de cette instruction est :



Exemple :

```
FOR ( instr1;cond1;instr2)
{ instr3;
  FOR (instr4;cond2;)
Instr5;}
```

- **L'instruction-générale** permet d'affecter à une variable une constante, une variable où bien le résultat d'une expression arithmétique (+, -, x , /).
- **L'instruction-condition** permet de comparer une variable avec une variable, où bien une variable avec une constante (exemple : x>12, Y>X).
- Une constante entière est une suite de chiffres. Elle peut être signée ou non signée tel que sa valeur est entre -32768 et 32767. Si la constante entière est signée, elle doit être mise entre parenthèses.
- Un identificateur est une suite alpha numérique qui commence par une lettre majuscule suivie d'une suite de chiffres et lettres minuscules. Un IDF ne doit pas contenir plus de 8 caractères.

1. Donner les expressions régulières nécessaire écrite en langage Flex. (1)
2. Donner les règles de traduction nécessaire écrite en langage Flex. (1.5)
   a. A quoi sert la table des symboles dans l'analyse syntaxico-sémantique? (0.5)
   b. En utilisant le langage Bison
      - Donner la grammaire l'expression ci-dessus (2), ainsi que le programme permettant de générer les quadruplets de **l'instruction FOR** dans le cas d'une analyse ascendante. Prendre en considération les imbrications d'instructions FOR. (2)
   c. Donnez le programme permettant de générer le code machine correspondant au code intermédiaire précédent sachant qu'on utilise la machine Assembleur 8086 composé de 4 registres. (1)

Les expressions régulières nécessaire écrite en langage Flex.

```
lettreM [A-Z]
lettre [a-zA-Z]
chiffre [0-9]
IDF {lettreM}({lettre}|{chiffre})*

cstEntiereNonSignee [0-9]+
cstEntiereSignee \([+-][0-9]+\)
```

Les règles de traduction nécessaire écrite en langage Flex.

```
FOR          {inserer(yytext, "mc");   yylval.strVal = strdup(yytext);    return mc_For;}

{idf}        {   if(yyleng <= 8)
                 {
                    yylval.strVal = strdup(yytext);
                    inserer(yytext, "idf");
                    return idf;
                 }
                 else
                    printf("Erreur lexicale a ligne %d IDF : %s trop long \n", nb_ligne, yytext);
return idf ;
             }

{cstEntiereNonSignee}    {Col= Col + strlen(yytext);
                          if ((atoi(yytext)>32768)||(atoi(yytext)<-32768))
         printf ("Erreur Lexical: Constante Entiere depasse la plage des valeurs a la ligne %d a la colonne %d
\n ",nb_ligne, Col);
         printf (" L entite reconnue est %s \n", yytext);
         return cstEntiereNonSignee
}
{cstEntiereSignee}      {Col= Col + strlen(yytext);
         if(atoi(yytext) >= -32768 && atoi(yytext) <= 32767)
              {
                 yylval.intVal = atoi(yytext);
                 inserer(yytext, "cst");
                 if(yytext[0] == '+' || yytext[0] == '-')
                    return entier_s;
                 else
                    return entier;
              }else
                 printf ("l'entier %s n'appartient pas a l'intervalle donne \n", yytext);
return cstEntiereSignee;
             }
"("    {Col= Col + strlen(yytext); printf (" L entite reconnue est %s \n", yytext); return parO;}
")"    {Col= Col + strlen(yytext); printf (" L entite reconnue est %s \n", yytext); return parF;}
";"    {Col= Col + strlen(yytext); printf (" L entite reconnue est %s \n", yytext); return pvg;}

< > =
```

```
[ \t]            Col= Col + strlen(yytext);
\n               {Col= 1; nb_ligne++;}
.                { printf ("Entite lexicale non reconnue a ligne %d a la colonne %d l entite %s \n", nb_ligne,
Col,yytext);}
```

La table des symboles dans l'analyse syntaxico-sémantique sert à :

La TS contient toutes les informations concernant les entités du programme. Elle est utile pour la détection les erreurs sémantiques.

La grammaire l'expression ci-dessus

```
FOR ( instr1;cond1;instr2)
{ instr3;
  FOR (instr4;cond2;)
Instr5;}
```

<**instruction-FOR**> **FOR** '(' <instruction-initialisation> ';' <instruction-condition> ';' ')' <instruction-générale> | **FOR** '(' <instruction-initialisation> ';' <instruction-condition> ' ;' <post-instruction> ')' <instruction-générale>

```
 list_INST : FOR list_INST
                    |
                    ;

FOR : mc_For parO  INST_Init pvg INST_cond  pvg INST_post parF  list_INST  pvg
;

INST_Init:  idf aff cst
          | idf aff EXP
;

INST_cond :  var comp var | var comp cst
;

INST_post: INST_Init
|
;
EXP:
;
```

Partie quadruplets: Décomposition de la grammaire

<instruction-FOR> <A> <B> **<instruction-générale>**

```
      {    Générer-quadruplet (BR, , , deb_for);
           qc++;
           QUAD(deb_cond1).4=qc;
       }
```

| <A> <C> <D> **<instruction-générale>**
{ Copier la Zone-Tampon dans la zone QUAD;
Générer-quadruplet (BR, , , **deb_for**);
qc++;
QUAD(**deb_cond2**).4=qc;
}

<A> FOR '(' **<instruction-initialisation>** ';' {**deb_for**=qc;}
<B> **<instruction-condition>**' ;' ')'
{ Générer-quadruplet (BZ, , ,);
deb_cond1=qc++;
}
<C> **<instruction-condition>** ';'
{ Générer-quadruplet (BZ, , ,);
deb_cond2=qc++;
dérouter la sortie des quadruplets vers Zone-Tampon
}
<D> **<post-instruction>** ')'
{Rediriger la sortie des quadruplets vers la zone de sortie normale QUAD; }

<instruction-initialisation> **IDF := Cst Quadr (:= ; $3 ; ; $1)**
| Idf := IDF Quadr (:= ; $3 ; ; $1)
;