Faculté Electronique et Informatique
Département Informatique
Module de Compilation2
1ère Année Master IL et MIV **Année Universitaire : 2015/2016**

EPREUVE DE COMPILATION

Exercice 1 : (7points)

- Donnez les différentes phases d'un compilateur en expliquant chacune d'elles.
- Donnez la définition d'une allocation statique et allocation dynamique.
- A quelles étapes de la compilation, nous parlons d'allocation mémoire.
- A quelles étapes de la compilatioh, nous parlons d'optimisation de code.
- De quoi a-t-on besoin pour générer un code correct et efficace.

Exercice 2 : (4points)

Considérons un langage L permettant d' écrire des codes sous la forme suivante :

Program ID
INT : ID
{IF (ID OPC CHIFFRE) ID = CHIFFRE ELSE ID =
CHIFFRE FIN
} ID

Avec :
- ID : suite de 5 lettre minuscules au maximum
- CHIFFRE : un eniter
- OPC : < , > , <= , >= , ==, !=

Donnez les codes permettant l'analyse lexicale, syntaxique, sémantique jusqu'à la génération des quadruplets, selon la syntaxe Flex et bison. Donnez seulement les entêtes des fonctions C utilisées pour la sémantique sans les implémenter.

Exercice 3 : (9points)

Soit le programme Algol suivant :

```
Program essai
L1: Begin  Integer I, L, n, array A[1:L; -1:2*L; 0:L+2];
     L2: Procédure Gamme(X1); X1: Integer;
          Begin Boolean S; Integer P, Array B[I:P];
               L3: Begin Real X, Y;
                         L4: X:=(X+((-P)/2))*(P-P/ Y);
                              .
                              .
                    L5: End;
          End;
     L6: Procédure Beta(X2); X2: Integer;
          Begin  Integer a,b, m, array C[-I.:I.], Val;
               L7 :  I:=0
                     a:=n-1
                     b:=m
                     t1:=5*a
                     I:=I+1
                     t2:=C[t1]
                     While I>L+6  Do K:=8*3,14
                                     I:=I+3
                                     D[I]:=K
                                  EndDo
                     t3:=5*a
                     z:=C[t3]
                     I:=I+1
                     J:=7*I
                              .
                              .
                              .
                    End;
                              .
                              .
                              .
I.8:End.
```

a-Donner Les différents états de la pile aux étiquettes L1, L2, L3, L4, L5, L6, L7, et L8.
b- Générer le code de l'instruction à L4 en utilisant les quadruplets et la machine vue en cours.
c- Optimiser le code dans la portion de programme à L7.

Corrigé Examen
de Compilation

Exercice 1 :

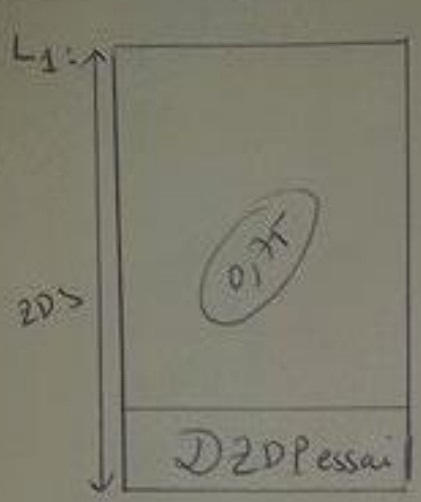
- Différents phases de la Compilation (Voir cours)
- Allocation statique et Allocation dynamique (Voir cours)
- Nous parlons d'Allocation mémoire pendant la Compilation, pendant la sémantique. Dans le cas statique, l'allocation mémoire effective - Dans le cas de langages à allocation dynamique, on génère du code qui réalise l'allocation effective se fait pendant l'exécution l'allocation permet d'obtenir une place et une adresse mémoire aux objets manipulés par le programme
- Nous parlons d'optimisation, à toutes les phases de la Compilation (du début à la fin), même pendant l'exécution (gain sur les choix des opérateurs, ...)
- Nous avons besoin d'une machine cible, avec bonne connaissance de son jeu d'instructions, ...

```
%{
#include <stdio.h>
#include <string.h>
#include "miniada.tab.h"
extern YYSTYPE yylval;
%}
%option yylineno
identificateur [a-z]{1,5}
chiffre 0|[1-9][0-9]*
integer {chiffre}
BLANC [ \t]+
Saut [\n]
%%
"INT"          { return INT; }
"Program"      { return PROGRAM;}
"IF"           { return IF;}
"DO"           { return DO;}
"ELSE"         { return ELSE;}
"FIN"          { return FIN;}
":"            { return ':';}
"="                          { return '=';}
"=="           { return EQ;}
">="           { return GE;}
"<="           { return LE;}
">"            { return G;}
"<"            { return L;}
"("                { return '(';}
")"                { return ')';}
"{"                { return '{';}
"}"                { return '}';}
{identificateur} { yylval.chaine=strdup(yytext); return ID;}
{integer}          { yylval.entier=atoi(yytext); return integer;}
{BLANC}        { }
{Saut}         { }
.              {printf (" erreur lexicale %d\n",yylineno);}
%%
int yywrap(){return 1; }
```

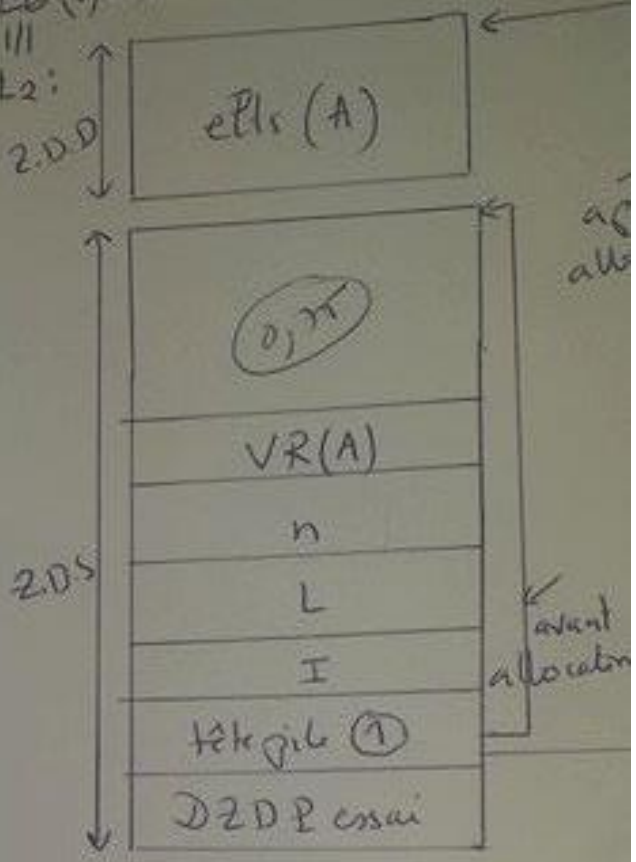
```
%union{
char *chaine;
int entier;
}

%token <chaine>ID
%token <entier> integer
%token PROGRAM INT IF ELSE DO FIN '(' ')' '(' ')' '=' ':'
%left G GE EQ DI LE L
%left '(' ')'
%%

S:PROGRAM ID Dec '{' code '}' ID { //vérifier que $2=$7
                                   printf("programme syntaxiquement juste\n");
};
Dec: INT ':' ID {//Si $3 existe dans la TS -> erreur double déclaration
                 //Sinon insertion de $1 dans la TS
};

code: p1 p2 p3;
p1: IF Comp {//Si $1 n'esiste pas dans la TS -> erreur IDF non déclaré
             //ajouter le quad de la comparaison)
;
p2: DO Affect {}
;
p3: ELSE Affect FIN {}
;
Affect : ID '=' integer {     //Si $1 n'esiste pas dans la TS -> erreur IDF non déclaré
                              //ajouter le quadruplet (=,$3," ",$1)}
%%

int yyerror(char* msg){printf("\n%s",msg);return 1;}

int main(){yyparse();return 0;}
```

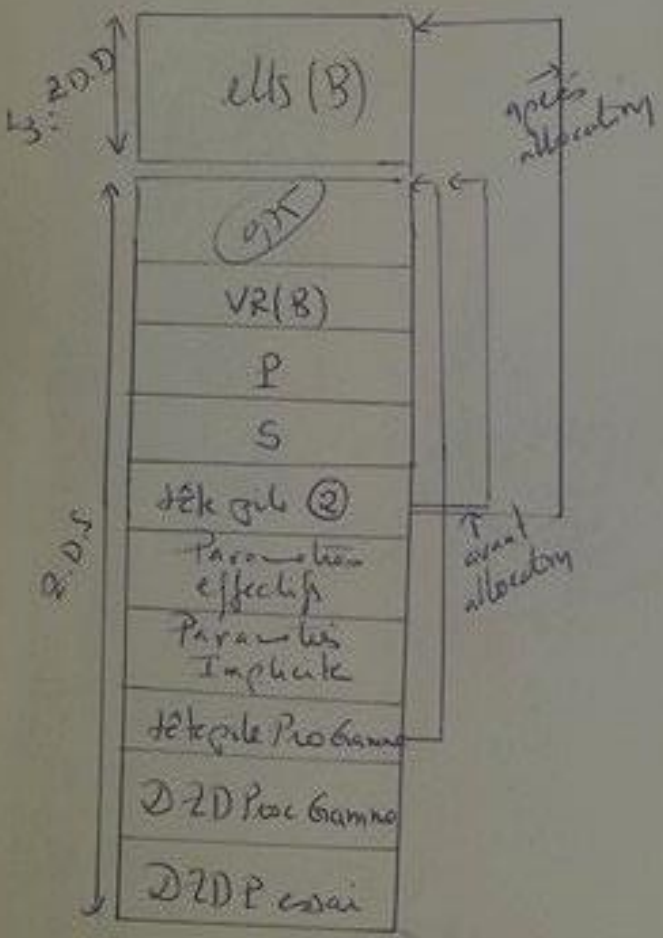
Exercice 3

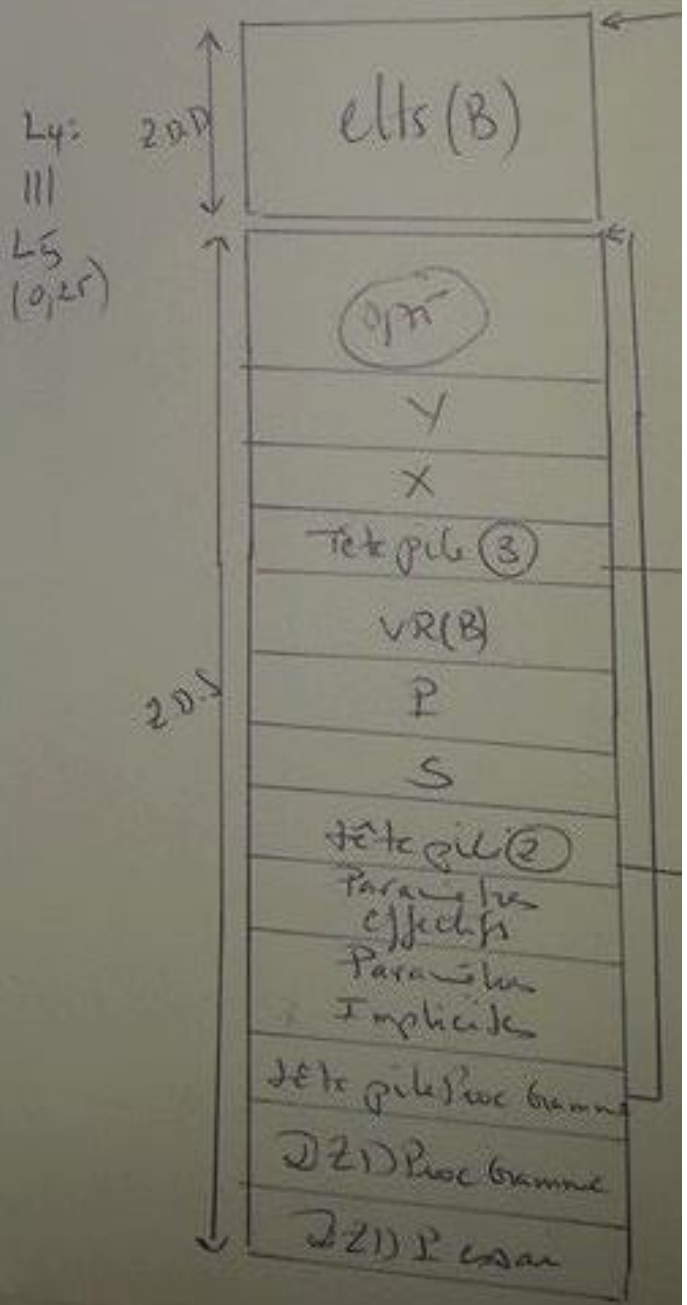
a)

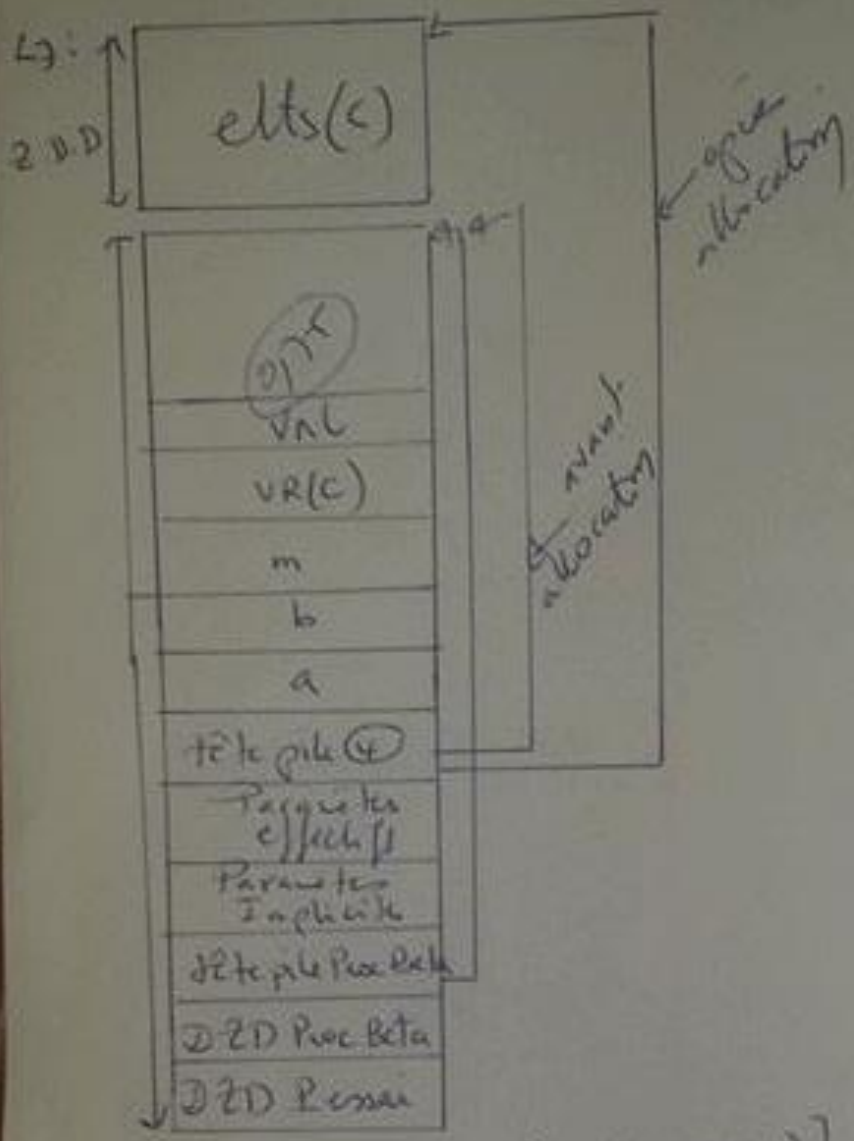


(0,2r) L8 ≡ L6 (0,2r) ≡ L2 :









b. $X := [(X + ((-P)/2)) * (P - (P/Y))]$

T_1 : $(-P)$
T_2 : $((-P)/2)$
T_3 : $(X + ((-P)/2))$
T_4 : (P/Y)
T_5 : $(P - (P/Y))$
T_6 : $[(X + ((-P)/2)) * (P - (P/Y))]$

Les quadruplets

1- (-, P, , T_1)
2- (/, T_1, 2, T_2)
3- (+, X, T_2, T_3)
4- (/, P, Y, T_4)
5- (-, P, T_4, T_5)
6- (*, T_3, T_5, T_6)
7- (:=, T_6, , X)

DJF

Get in Acc	Acc	code
	vide	
Get in Acc('P', '1')	P	Load P
	T_1	CHS
Get in Acc('T_1', '1')	T_2	DIV 2
Get in Acc ('X', 'T_2')		
Factorisation des opérandes dans le quadruplet (+, T_2, X, T_3)	T_3	ADD X
		Store T_3
Get in Acc ('P', '1')	P	Load P
	T_4	Div Y
Get in Acc ('P', '1')		Store T_4
	P	Load P
	T_5	SUB T_4
Get in Acc ('T_3', 'T_5')		
Factorisation des opérandes dans le quadruplet (x, T_5, T_3, T_6)	T_6	MULT T_3
		Store X

c) Optimisation :

I := 0
a := n-1
b := m
t_1 := 5*a
I := I+1
t_2 := C[t_1]
h := L+6
K := 8 * 3,14

while I > h DO I := I + 3
D[I] := K
EndDO
J := 7*(I+1)

(2)