Examen de compilation
Durée lh30

Exercice 1(8 pts) :

Soit le langage des instructions 'switch' sous le langage C.

1. Donner G(L).

V_N = { S, EXP, CASES, BREAK, DEFAULT, CSTE, INSTRS }
V_T = {switch, case, break; , default, : , printf("addition"); , printf("soustraction"); , printf("Error!"); , operation, '+', '-' , { , } , (,) }

S ⟶ switch (**EXP**) { **CASES DEFAULT** }
CASES → case **CSTE** : **INSTRS BREAK CASES** | ϵ
BREAK → break; | ϵ
INSTRS → printf("addition"); | printf("soustraction"); | printf("Error!"); | … | **INSTRS** | ϵ
DEFAULT → default : **INSTRS** | ϵ
EXP → operation
CSTE → '+' | '-' | …

2. Soit le mot w de L :

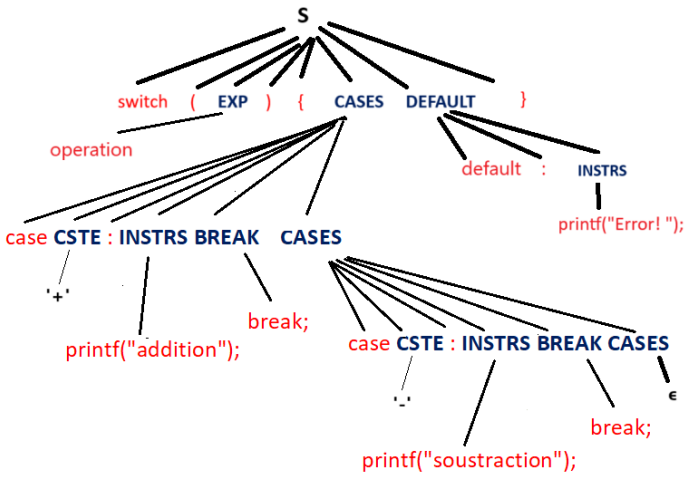
```
switch(operation)
{
        case '+' :
                printf("La somme");
                break;
        case '-' :
                printf("soustraction");
                break;
        default:
                printf("Error! ");
}
```

a) Donner la dérivation du mot avec la méthode LR.

Tampon	Pile	Opération
$switch(operation) {case '+' : printf("La somme"); break; case '-' : printf("soustraction");break; default:printf("Error! ");}	$	D + D + D
$) {case '+' : printf("La somme"); break; case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(operation	D
$) {case '+' : printf("La somme"); break; case '-' : printf("soustraction");break;	$switch(operation	R : EXP → operation

default:printf("Error! ");}		
$: printf("La somme"); break; case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP	D + D + D + D
$: printf("La somme"); break; case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){case '+'	R : CSTE → '+'
$break; case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){case CSTE	D + D
$break; case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){case CSTE: printf("La somme");	R : INSTRS → printf("La somme") ;
$break; case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){case CSTE: INSTRS	D
$case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){case CSTE: INSTRS break;	R : BREAK → break;
$case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){case CSTE: INSTRS BREAK	R : CASES → ϵ
$case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){case CSTE: INSTRS BREAK CASES	R : CASES → case CSTE: INSTRS BREAK CASES
$case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){ CASES	R : CASES → ϵ
$case '-' : printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){	D + D
$: printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){ case '-'	R : CSTE → '-'
$: printf("soustraction");break; default:printf("Error! ");}	$switch(EXP){ case CSTE	D + D
$break; default:printf("Error! ");}	$switch(EXP){ case CSTE: printf("soustraction");	R : INSTRS → printf("soustraction");
$break; default:printf("Error! ");}	$switch(EXP){ case CSTE: INSTRS	D
$default:printf("Error! ");}	$switch(EXP){ case CSTE: INSTRS break;	R : BREAK → break;
$default:printf("Error! ");}	$switch(EXP){ case CSTE: INSTRS BREAK	R : CASES → ϵ
$default:printf("Error! ");}	$switch(EXP){ case CSTE: INSTRS BREAK CASES	R : CASES → case CSTE: INSTRS BREAK CASES
$default:printf("Error! ");}	$switch(EXP){ CASES	D + D + D
$}	$switch(EXP){ CASES default : printf("Error! ");	R : INSTRS → printf("Error!");
$ }	$switch(EXP){ CASES default : INSTRS	R: DEFAULT → default : INSTRS
$ }	$switch(EXP){ CASES DEFAULT	D
$	$switch(EXP){ CASES DEFAULT }	R : S → switch (EXP) { CASES DEFAULT }
$	S	**ACCEPTE**

b) Donner la dérivation du mot avec la méthode LL.



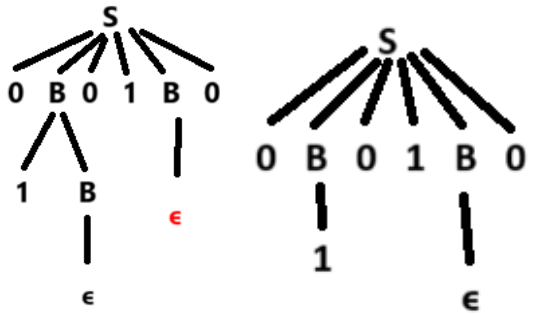
3. Soit la grammaire G(L) définit par :

S ⟶ 0B01B0

B ⟶ 0B|1B|0|1|epsilon

a) Prouver que la grammaire est ambiguë.

Pour le mot : 01010 Nous avons deux arbres de dérivation. Donc la grammaire est ambigüe.



b) Donner la dérivation des mots suivants 0101010 avec la méthode LR.

Tampon	Pile	Opération
$0101010	$	D
$101010	$0	D
$01010	$01	R : B → epsilon
$01010	$0B	D
$1010	$0B0	D
$010	$0B01	D
$10	$0B010	D
$0	$0B0101	R : B → 1
$0	$0B010B	R : B → 0B
$0	$0B01B	D
$	$0B01B0	S → 0B01B0
$	$S	Accepté

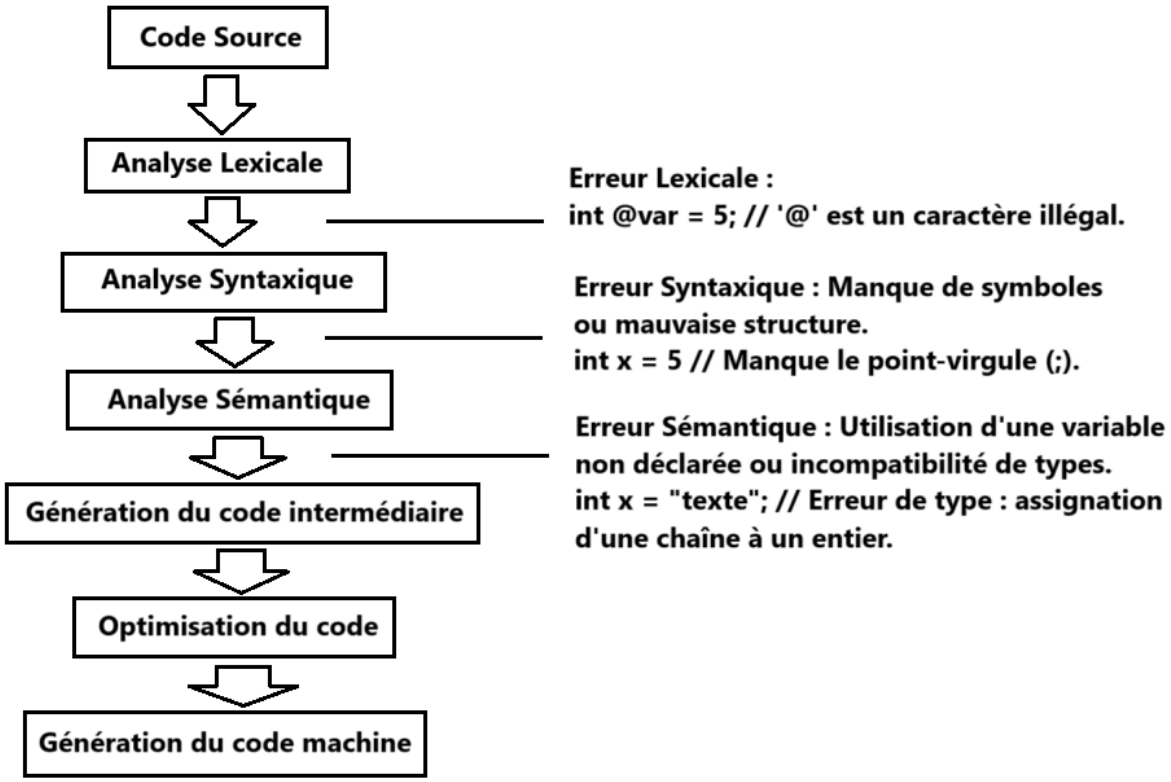
c) En déduire l'expression régulière de L.

0 (0|1)* 01 (0|1)* 0

Exercice II (12 pts)

Soit le langage L sur Σ={a,b,c} des mots qui comportent un nombre pair de 'b' et un **nombre pair** de 'a' ou bien un nombre pair de 'b' **et nombre impair** de 'a'.

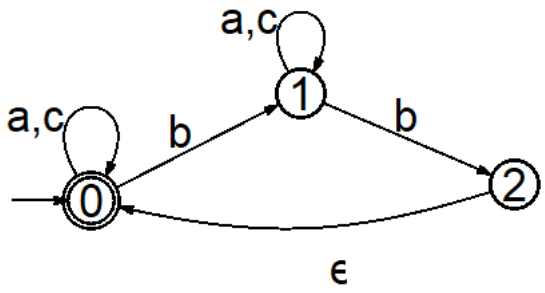
1. Schématiser les phases d'un compilateur. Pour chaque étape d'analyse donner un exemple d'erreur détectée.



2. Donner l'expression régulière de L.

e(L) = $[\ [b(a+c)^*b]^*(a+c)^*\ ]^*$

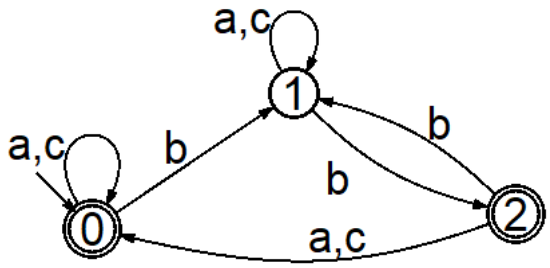
3. Donner l'AFN qui engendre L.



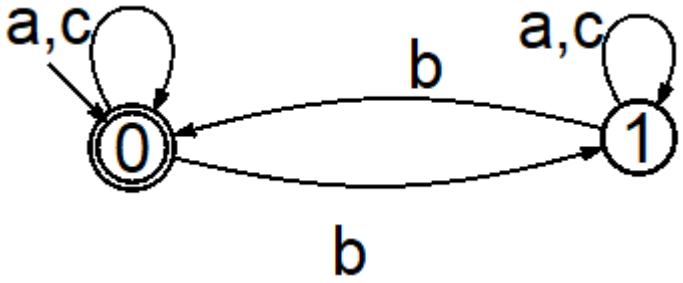
NB : Il est clair que les deux état 0 et 2 sont équivalents, on peut les éliminer dès maintenant. Si non nous pouvons continuer les détecter dans la minisation.

4. En déduire l'AFD de L.

	a	b	c
{0}	{0}	{1}	{0}
{1}	{1}	{0,2}	{1}
{0,2}	{0}	{1}	{0}



5. En déduire l'AFDM de L.



6. En déduire la matrice de transition de L.

	a	b	c
0	0	1	0
1	1	0	1

7. Donner la fonction int trace(char * mot, int **M, int taille) qui permet de calculer la trace d'un mot.

```
int trace(char *mot, int** M, int *T, int taille){
        int id = mot[0]-97; // convertir a en 0 et b en 1 pour les colonnes
        int tr = M[0][id];
        int i=1;
         while(i<taille){
                id = mot[i]-97;
                tr = M[tr][id];
                i++;
```

```
        }
        return tr;
}
```