Université Cadi Ayyad Marrakech
Faculté des sciences du travail
Département informatique

ZITOUNI HAMZA

TP ANALYSE LEXICALE

Début {
ma_Varaibale = 12;
b = 2;
si(var1!=b)alors{
var1 = b;
}
fin
	Code 1

A. L'analyse lexical

1- Quels sont les mots clé de ce langage ?

Debut
ma_variable , b , var1
=
12 , 2
;
si
(
!=
)
alors
{
}
fin

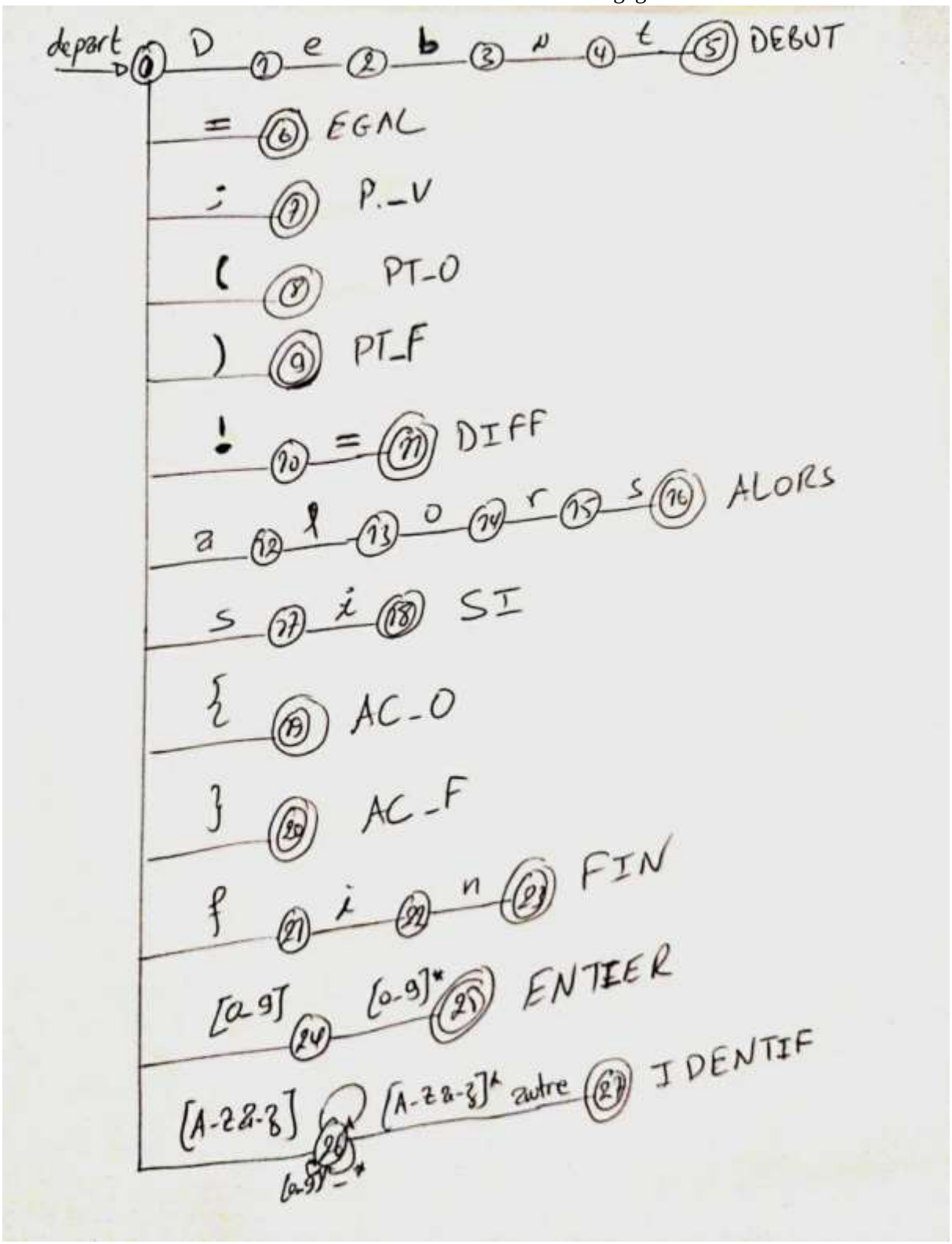
2- Définir les tokens correspondants aux mots clé de la question 1.

```
debut        DEBUT
ma_variable , b , var1  IDENTIF
=            EGAL
12 , 2       ENTIER
;            PT_V
si           SI
(            PT_O
)            PT_F
!=           DIFF
alors        ALORS
}            AC_O
}            A_F
fin          FIN
```

ZITOUNI HAMZA

TP ANALYSE LEXICALE

3- Donner les automates reconnaissants les tokens dans ce langage



4- Donner les expressions régulières qui permettent de définir les tokens de ce langage

nbr [0-9]
entier {nbr}+
identif [a-zA-Z_][a-zA-Z_0-9]*

C. Création d'un analyseur lexical

1-2 – Fichier UNITESLEXICALES.h :

uniteLexicales.h × fcb.bat

C: > Users > HP > Desktop > ● > compilation >

```
1
2   #define          DEBUT   257
3   #define          IDENTIF 258
4   #define          EGAL    264
5   #define          ENTIER  285
6   #define          PT_V    280
7   #define          SI      265
8   #define          PT_O    266
9   #define          PT_F    298
10  #define          DIFF    299
11  #define          ALORS   300
12  #define          AC_O    301
13  #define          A_F     302
14  #define          FIN     303
15
16  extern int valEntier;
17  extern char valIdentif[];
```

ZITOUNI HAMZA

TP ANALYSE LEXICALE

3- Créer un fichier .l et recopier le **Code 2.** 4- Créer un fichier .c et recopie le **Code 3.**

lexical.l

D: > s6 > compilation > tp1 > lexical.l

```
%{
#include <string.h>
#include "unitesLexicales.h"
%}
nbr [0-9]
entier {nbr}+
identif [a-zA-Z_][0-9a-zA-Z_]*
%%
debut { ECHO; return DEBUT; }
fin { ECHO; return FIN; }
{entier} { ECHO; valEntier = atoi(yytext); return ENTIER; };
{identif} {ECHO; strcpy(valIdentif, yytext); return IDENTIF; }
. { ECHO; return yytext[0]; }
%%
int valEntier;
char valIdentif[256];
int yywrap(void) {
return 1;
}
```

4- Créer un fichier .c et recopie le **Code 3.**

main.c 2

D: > s6 > compilation > tp1 > main.c > main(void)

```
/*Mon Fichier principal .c */
#include <stdio.h>
#include "unitesLexicales.h"
int main(void) {
int unite;
do {
unite = yylex();
printf(" (unite: %d", unite);
if (unite == ENTIER)
printf(" val Entier: %d", valEntier);
else if (unite == IDENTIF)
printf(" Nom Identif : '%s'", valIdentif);
printf(")\n");
} while (unite != 0);
return 0;
}
```

Université Cadi Ayyad Marrakech
Faculté des sciences du travail
Département informatique

ZITOUNI HAMZA

TP ANALYSE LEXICALE

5- Lancer l'invite de commande et taper les commandes suivantes pour générer l'analyseur lexical

```
flex -omonCompilateur.c lexical.l
gcc monCompilateur.c main.c -o prog
./prog<test.txt
```

6- Recopier le code du programme **Code 1** dans fichier .txt et taper la commande suivante pour tester votre analyseur

D: > s6 > compilation > tp1 > text.txt

```
Debut
ma_Varaibale=12; b=2;
si(var1!=b) alors{
    var1 = b;
}
fin
```

Après l'exécution de toutes les commandes, on a trouvé le résultat suivant :

```
D (unite: 264 Nom Identif :D
 (unite: -61
 (unite: -87
but (unite: 264 Nom Identif :but

ma_Varaibale (unite: 264 Nom Identif :ma_Varaibale
= (unite: 61
12 (unite: 259 val Entier: 12
; (unite: 59

b (unite: 264 Nom Identif :b
= (unite: 61
2 (unite: 259 val Entier: 2
; (unite: 59

si (unite: 264 Nom Identif :si
( (unite: 40
var1 (unite: 264 Nom Identif :var1
! (unite: 33
= (unite: 61
b (unite: 264 Nom Identif :b
) (unite: 41
  (unite: 32
alors (unite: 264 Nom Identif :alors
{ (unite: 123

var1 (unite: 264 Nom Identif :var1
  (unite: 32
= (unite: 61
  (unite: 32
b (unite: 264 Nom Identif :b
; (unite: 59

} (unite: 125

fin (unite: 258
 (unite: 0
```