UNIVERSITE ABDERRAHMANE MIRA DE BEJAIA

FACULTE DES SCIENCES EXACTES

Département d'Informatique

Rapport :

TP Théorie des langages

Réalisé par :

Seddar Abdelhadi

Encadré par :

Mme. Zamouche

But de cet Rapport :

C'est de Créer un programme Qui Vérifie Si un mot appartient à un langage en utilisant son automate

Index

I) Langage a Etudier

On Prend le langage Suivant : dans l'alphabet {a,b}

$$\mathbf{L = \{ ba^n / n \geq 0 \}}$$

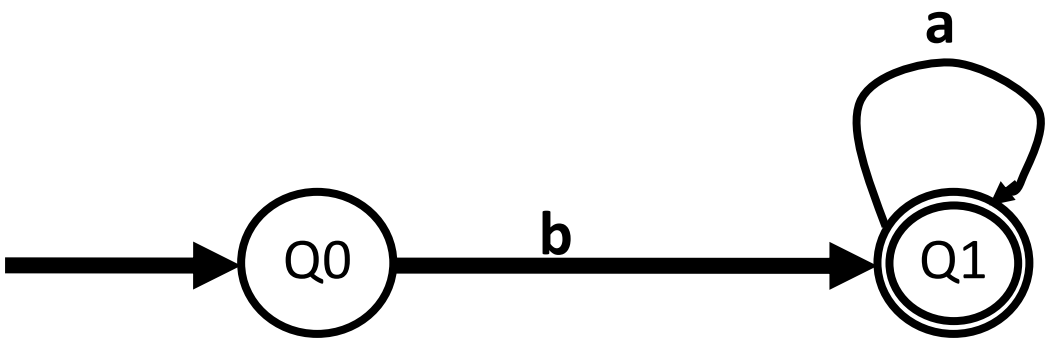
On déduit la grammaire suivante :

L: G = ({a, b} , {S, A} , P, S }

P: **S** → b**A**

A → a**A** | ε

Et ce qui génère l'automate Suivant :



II) Code Program

A) Introduction :

Ce program est programmé en langage C en utilisant la récursivité et la logique des listes chainées.

B) Déclarations:

On va mettre le code relatif à la déclaration de notre automate dans Deux Fichiers _.h et _.c

1) Dans le Fichier _.h
   - **a.** on vas définir une structure **Q** (**struct AUTOMATE_T**) Comme Suivant

```
30  typedef struct AUTOMATE_T
31  {
32      int n_Next;
33
34      char *c;
35      int *Next;
36  } Q;
```

Int n_Next → Nombre de prochaine Etats possibles
Char *c → Un Tableau de taille [n_Next] qui sauvegarde la lettre qu'on peut obtenir dans la prochaine état suivante
Int *Next → Un Tableau de taille (n_Next) qui va stocker l'index des Etats Suivants possible. (On va l'allouer dans le fichier _.c)

Note :
L'utilisation du « Next » et « c » sera expliqué ultérieurement

   - **b.** On va définir les fonctions Suivantes :

```
52  extern void NewQs();
53  extern int ExistNextC(int _Index_Current, char _Character, int *_Index_Next);
54  extern int CheckWord(char *__Word);
```

NewQs () → Pour Créer Toutes les états d'automate.
Retourne : NULL (il remplit le tableau q (voir « Dans fichier_.c »))

ExistsNextC () → Chercher dans Si le caractère le **_Character** peut être obtenu avec l'une des prochaine états du l'Etat[**_Index_Current**] et retourner l'index du prochain état qui le contient.

Retourne : 0 Il n'existe pas un état qui contient la lettre **_Character**.
1 Il existe Etat qui contient la lettre **_Character**.

CheckWord () → Vérifier si le mot appartient ou non au langage

Retourne : -1 Mot vide.
0 Mot Appartient pas.
1 Mot Appartient.

2) Dans le Fichier _.c

```
3   Q *q = NULL;
4
5   int __N_Etas_Debut = 1;
6   int *__Etas_Debut;
7
8   int __Etas;
9   char *__Langage;
```

Q *q → C'est un Tableau de Type Q (**struct AUTOMATE_T**) qui sauvegarde Tout les états et leur informations.

Valeur par default : NULL (**Tableau pas définis ni Alloué de la mémoire**)

Int _N_Etas_Debut → Un entier qui sauvegarde Combien d'états l'automate peut commencer avec.

Valeur par default : 1

Int *_ Etas_Debut → C'est un Tableau de Type Int (**Entier**) qui sauvegarde Combien d'états l'automate peut commencer avec.

Valeur par default : NULL

Int _Etas → Un entier qui sauvegarde Combien d'états l'automate a.

Valeur par default : 0

Char *_ Langage → C'est une chaine de caractères qui sauvegarde le langage utilisé pour créer l'automate (utilisé sauf a l'affichage)

Valeur par default : NULL

C) Définitions:

On va mettre le code relatif à notre automate dans le Fichier _.c :

Dans le fichier _.h on a Déclaré juste 3 fonctions pour qu'on puisse les utilise dans d'autres fichiers comme main.c (Ou se trouve l'emplacement de la fonction principale) ou de n'importe quel autre fichier. On a juste besoin d'inclure le fichier _.h

Par exemple dans mon code : `#include "automate/_.h"` le fichier _.h dans le dossier automate.

Les fonctions définis dans le fichier _.c Sont :

NewQ() → Pour Créer Un Nouveaux état
Retourne : Le Nouveaux état

NewQs() → Voir b.1) Dans le Fichier _.h

ExistsNextC () → Voir b.1) Dans le Fichier _.h

CheckWord_R () → Une fonction récursive qui Vérifie si le mot appartient ou non au langage choisie, « Sert comme une extension pour CheckWord() »
Retourne : 0 Mot Appartient pas.
1 Mot Appartient.

CheckWord () → Voir b.1) Dans le Fichier _.h

1) NewQ() :

```
11  Q NewQ(char *__Characteres_Possibles, int __Nombre_Possibilites)
12  {
13      Q q; 1
14      q.n_Next = __Nombre_Possibilites; 2
15
16      q.c = calloc(__Nombre_Possibilites, sizeof(char));
17      q.Next = calloc(__Nombre_Possibilites, sizeof(int)); } 3
18
19      strncpy(q.c, __Characteres_Possibles, __Nombre_Possibilites); 4
20      return q; 5
21  }
```

1- Déclarer une variable **q** de type Q (**struct AUTOMATE_T**) ou on vas stocker notre nouvelle état
2- Copier le nombre d'état prochain possible (**_Nombre_Possibilites**) vers le n_Next du variable **q**.

3- Allouer « **q.n_Next** » d'espace pour le tableau **q.c** et **q.Next**
4- Copier les « **q.n_Next** » valeurs des **_Charctere_Possibles** vers **q.c**
5- Retourner le **q**

2) NewQs() :

```
31  void NewQs()
32  {
33      __Langage = "L = {ba^(n) / n >= 0}"; 1
34
35      __Etas = 2; 2
36      q = calloc(__Etas, sizeof(Q)); 3
37
38  5   q[0] = NewQ("b", 1);
39      q[0].Next[0] = 1;
40
41      q[1] = NewQ("\0a", 2);
42  6   q[1].Next[0] = _e_;
43      q[1].Next[1] = 1;
44
45      __N_Etas_Debut = 1;
46  7   __Etas_Debut = calloc(__N_Etas_Debut, sizeof(int));
47
48      __Etas_Debut[0] = 0; 8
49  }
```

1- Ecrire le langage qui peut être lu par l'utilisateur.
2- Nombre d'états.
3- Allouer dans la mémoire « **__Etas** » cases de type Q
4- L'espace pour définir chaque état

Note :

Pour Chaque êtas on doit suive certains instructions avec un ordre spécifié

1- l'êtas **n** on doit l'écrire dans **q[n]** (Veillez démarrer de 0)
2- il faut conserver l'ordre de l'état et le Caractère qui correspond

```
q[n] = NewQ("abc...", m);
q[n].Next[0] = n1;
q[n].Next[1] = n2;
.
.
.
q[n].Next[0] = nm;
```

Exemple :

On a n2 qui génère le char 'b' dans NewQ on vas écrire b en 2ème et écrire l'index du n2 dans La 2ème case du tableau **q[n].Next**

Cas Spécial :

1- En cas la lettre générée est ε on écrit la même valeur du Macros _v_e_

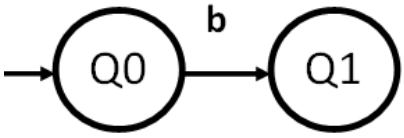
2- En cas l'état est terminal on doit ajouter le prochain (Ajouter +1 s'il n'a pas été inclus dans le nombre des êtas prochain) Et écrire dans la dernière case de q[n].Next l'index de ε (Ecrire la la valeur du macro _e_

Valeur par default des macros:

_v_e_ → '\0'

_e_ → -1

5- Dans mon automate j'ai l'état Q0 qui est définis comme suit



Alors les caractères sont 'b' (1 caractère) et a un seul Etas prochain qui est Q1

6- Dans mon automate j'ai l'état Q1 qui est définis comme suit

Q1 est un état terminal Alors les caractères sont ε ('\0') et 'a' (2 caractères) et pour chaque caractère a un état suivant

ε → **index d' ε (**_e_**)**

Q1 → **Index de lui-même (**Q1**)**

3) ExistNextC() :

```
51  int ExistNextC(int _Current_Index, char _Character, int *_Index_Next)
52  {
53      for (int i = 0; i < q[_Current_Index].n_Next; i++)
54      {
55          if (q[_Current_Index].c[i] == _Character)
56          {
57              if (_Index_Next != NULL)
58                  *_Index_Next = i;
59              return 1;
60          }
61      }
62      return 0; 4
63  }
```

2 3 1

1- Une boucle pour vérifier dans le tableau q[**_Current_Index**].**Next** de 0 jusqu'à n_next -1 (il peut ne pas arriver a la fin si il trouve un état qui génère le caractère **_Character** (voir C.3) 3-))
2- Vérifier s'il existe un état qui génère le caractère **_Character** si oui on vas aller à 3-
3- on va retourner l'index ou **_Character** est généré puis retourner 1 et quitter la boucle et la fonction.

Note :
Si tu veut pas retourner la valeur d'index qui contient celui cherché dans l'appel met au 3ème argument la valeur **NULL** (voir C.4) 2-)

En cas le programme est 'arrivé a cet instruction veut dire qu'il n a pas trouvé un état qui génère le caractère cherché et retourner 0.

4) CheckWord() :

```
78  int CheckWord(char *_Word)
79  {
80      if (!strcmp(_Word, "\n") || !strcmp(_Word, "\0"))
81      {
82          for (int u = 0; u < __N_Etas_Debut; u++)
83              if (ExistNextC(__Etas_Debut[u], _v_e_, NULL))
84                  return 1;
85          return -1;
86      }
87
88      int i, re = 0;
89
90      for (int u = 0; u < __N_Etas_Debut; u++)
91          if (ExistNextC(__Etas_Debut[u], _Word[0], &i))
92          {
93              re = CheckWord_R(_Word + 1, q[0].Next[i], &i);
94              if(re)
95                  return re;
96          }
97      return 0;
98  }
```

1- Vérifier si le mot lu est vide ou non, Si vrai on va à 2-
2- Vérifier si l'une des états dans _Etas_Debut génère ε
   Si **oui** → retourner 1
   Si **non** → retourner -1
3- Vérifier dans tous les états de départ si le mot existe
4- Vérifier l'état s'il génère la première lettre du mot **_Word** si
   Si **oui** → Aller a 5-
   Si **non** → retourner a 4- et vérifier la prochaine êtas sinon quitter la boucle.

5- (voir C.5))Vérifier la prochaine lettre du mot **_Word** et sauvegarder ce qu'il retourne dans la variable **re**. Apres si :

Si **re** est égal a 1→ retourner 1

Sinon → Continuer la boucle si possible

6- Si la boucle a terminé veut dire le mot n'appartient pas au langage

5) CheckWord_R() :

```
65  int CheckWord_R(char *__Word, int __Current_Index, int *_Store_Index)
66  {
67      if ((!strcmp(__Word, "\n") || !strcmp(__Word, "\0")))
68          if (ExistNextC(__Current_Index, _v_e_, NULL))
69  1           return 1;
70          else
71              return 0;
72
73      if (ExistNextC(__Current_Index, __Word[0], _Store_Index))
74      {
75  2       return CheckWord_R(__Word + 1, q[__Current_Index].Next[*_Store_Index], _Store_Index); 3
76      }
77      return 0;4
78  }
```

1- Vérifier si le mot est vide puis si l'état indexé par _Current_Index génère ε

Si **oui** → retourner 1

2- Vérifier la première lettre du mot **_Word** a un Etas qui génère ce dernier

Si **oui** → aller à 3-

Si **non** → aller à 4-

3- Vérifier le mot **_Word** sauf sa première lettre en appelant CheckWord_R()

Et changer _Current_Index vers l'index prochains (**q[_Current_Index].Next[*_Store_Index]**)

4- Si le programme est arrivé a cet instruction, ca veut dire que le mot n'appartient pas car il 'y a pas un état qui génère la lettre **_Word[_Current_Index]**

D) Fonction principale:

Les instructions écrite dans le programme principale, peut être écris dons n'importe quel fonction dans votre program pour l'utiliser.

1) Lecture du mot :

Vous avez 2 choix :

- avec scanf simple mais vous pouvez pas lire des mots vides
- avec getline pas compliqué mais la fin du mot sera \n\0 au contraire du scanf qui termine les mots lus avec \0

Mon programme marche avec les deux, tant que avec le scanf est simple j'ai utilisé getline pour expliquer comment l'utiliser.

D'abord on doit déclarer 2 variables :

```
6   char *mot;
7   size_t s_mot;
```

Char *mot → ou stocker la chaine de Caractère
Size_t s_mot → Taille maximum de la chaine de caractère a lire.

Dans la fonction principale on doit allouer l'espace pour le mot a lire

```
7       s_mot = 1024;
8       mot = malloc(s_mot * sizeof(char));
9       char *re;
```

Puis utiliser getline()

```
19
20      getline(&mot, &s_mot, stdin);
```

Après la lecture du mot on vas vérifier le mot puis écrire si le mot appartiens ou pas

```
switch (CheckWord(mot))
{
case -1:
    re = MOT_VIDE;
    break;
case 0:
    re = MOT_NO;
    break;
case 1:
    re = MOT_YES;
    break;
default:
    re = "Erreur\n";
    break;
}
```

On vas mettre le texte a écrire dan une variable **re** (de type **char *)**

Note :

Valeur par default des macros:

MOT_VIDE → "Mot Vide."
MOT_NO → "Mot Appartient pas."
MOT_YES → "Mot Appartient."

III) Conclusion

Dans ce projet j'ai créer un programme qui Vérifie si un mot appartient dans un certain langage en utilisant son automate, Cet programme est un peu long, mais avec lui on peut changer le langage en utilisant son automate bien-sur en changeant la fonction NewQs() mais en appliquant les règles spécifié dans l'explication de cette fonction.