Université Mohamed Premier
Faculté des Sciences
Oujda

Filière SMI - Semestre 5
Module : Compliation
Année universitaire $2016 - 2017$

Solutions des TP N° 1

Exercice 1

1. Un reconnaisseur en C pour le langage des chaînes qui commençent par 0 et se terminent par 11 :

   ▷ Tout d'abord, il faut vérifier que le premier caractère lu est un '0', sinon on retournera la valeur 0.

   ▷ Ensuite, pour les autres caractères, il faut vérifier que chacun d'eux est dans l'ensemble $\{0,\ 1,\ \backslash n\}$.

   ▷ Il faut chercher une séquence de la forme '1', '1' '$\backslash n$'.

   ► **Algorithme** :

```
Fonction AnaLex() : Booléen;
   Variables :
      car : Caractère;
      indicateur : Booléen;
   Début
      indicateur := Faux;
      lire(car);
      Si(car != '0')
         Alors Retourner Faux;
      FinSi
      Répéter
         lire(car);
         Si(car = '0')
            Alors indicateur = Faux;
         Sinon Si(car = '1')
                  lire(car);
                  Si(car = '0')
                     Alors indicateur = Faux;
                  Sinon Si(car = '1')
                           Alors indicateur = Vrai;
                        FinSi
```

```
                FinSi
         FinSi
      Jusqu'à (car = '\n' ou (car != '0' et car != '1'));
      Si(car != '\n')
         Alors Retourner Faux;
      Sinon
         Retourner indicateur;
      FinSi
   Fin
```

▶ **Programme** C :

```
#include <stdio.h>
int AnaLex()
{
   char car;
   int indicateur = 0;

   car = getchar();
   if(car != '0')
     return 0;
   do
   {
      car = getchar();
      if(car == '0')
         indicateur = 0;
      else
      {
         if(car == '1')
         {
            car = getchar();
            if(car == '0')
              indicateur = 0;
            else
            {
               if(car == '1')
                   indicateur = 1;
            }
         }
      }
   }while(car != '\n' && (car == '0' || car == '1'));
   if(car != '\n')
     return 0;
   else
```

```
     return indicateur;
}

int main()
{
   if(AnaLex())
     printf("Chaine acceptee\n");
   else
     printf("Chaine non acceptee\n");
   system("pause");
   return 0;
}
```

2. Un reconnaisseur en C pour le langage des chaînes ayant un nombre pair de 0 :

▶ **Algorithme** :

```
Fonction AnaLex() : Booléen;
   Variables :
      car : Caractère;
      pair : Booléen;
   Début
      pair := Vrai;
      Répéter
         lire(car);
         Si(car = '0')
            Alors pair = Non pair;
         FinSi
      Jusqu'à (car = '\n' ou (car != '0' et car != '1'));
      Si(car != '\n')
         Alors Retourner Faux;
      Sinon
         Retourner pair;
      FinSi
   Fin
```

▶ **Programme** :

```
#include <stdio.h>

int AnaLex()
{
   char car;
   int pair = 1;
```

```
   do
   {
      car = getchar();
      if(car == '0')
         pair = ! pair;
   }while(car != '\n' && (car == '0' || car == '1'));

   if(car != '\n')
     return 0;
   else
     return pair;
}
```

3. Un reconnaisseur en C pour le langage des chaînes ayant exactement un seul 1 :

▶ **Algorithme** :

```
Fonction AnaLex() : Booléen;
   Variables :
      car : Caractère;
      nb_un : Entier;
   Début
      nb_un := 0;
      Répéter
         lire(car);
         Si(car = '1')
            Alors nb_un := nb_un + 1;
         FinSi
      Jusqu'à (car = '\n' ou (car != '0' et car != '1'));
      Si(car != '\n')
         Alors Retourner Faux;
      Sinon
         Retourner (nb_un = 1);
      FinSi
   Fin
```

▶ **Programme** :

```
#include <stdio.h>

int AnaLex()
{
   char car;
   int nb_un = 0;
```

```
   do
   {
      car = getchar();
      if(car == '1')
         nb_un++;
   }while(car != '\n' && (car == '0' || car == '1'));

   if(car != '\n')
     return 0;
   else
     return (nb_un == 1);
}
```

4. Un reconnaisseur en C pour le langage des chaînes w pour lesquelles le nombre $2|w|_0 + |w|_1$ est divisible par 3 :

   ▶ **Algorithme** :

```
Fonction AnaLex() : Booléen;
   Variables :
      car : Caractère;
      nb_zero, nb_un : Entier;
   Début
      nb_zero := 0;
      nb_un := 0;
      Répéter
         lire(car);
         Si(car = '1')
            Alors nb_un := nb_un + 1;
            Sinon Si(car = '0')
                    Alors nb_zero := nb_zero + 1;
                  FinSi
         FinSi
      Jusqu'à (car = '\n' ou (car != '0' et car != '1'));
      Si(car != '\n')
         Alors Retourner Faux;
      Sinon
         Retourner ((nb_un + 2 * nb_zero) mod 3 = 0);
      FinSi
   Fin
```

   ▶ **Programme** :

```
#include <stdio.h>
```

```
int AnaLex()
{
   char car;
   int n0 = 0; /* nombre des 0 */
   int n1 = 0; /* nombre des 1 */

   do
   {
      car = getchar();
      if(car == '0')
         n0++;
      else
      {
         if(car == '1')
           n1++;
      }
   }while(car != '\n' && (car == '0' || car == '1'));

   if(car != '\n')
     return 0;
   else
     return ((2 * n0 + n1) % 3 == 0);
}
```

Remarque : la fonction "main" est la même pour tous les programmes !

Exercice 2

Il suffit de traduire en C l'algorithme de reconnaissance d'un mot par un **AFD** complet vu en cours. Il faut tout d'abord fixer une notation pour reptésenter les éléments d'un **AFD**. Voici une notation simple :

- ▷ L'ensemble des états E : s'il y a N états, on les note de 0 à $N-1$.
- ▷ L'alphabet A : un tableau A indexé de 0 à $K-1$, où K est la taille de l'alphabet.
- ▷ L'état initial q_0 : l'état 0.
- ▷ La fonction de transition δ : une matirice $M[i,j] = \delta(i,\ j)$, où $i = 0, 1, ..., N-1$ désigne un état et $j = 0, 1, ..., K-1$ désigne un symbole de l'entrée.

▷ L'ensemble des états finaux F : un tableau d'entiers F indexé de 0 à $N_f - 1$ où N_f est le nombre des états finaux.

On utilisera une fonction qui renvoie l'indice d'un symbole (dans le tableau qui représente l'alphabet). En fait, la notation du cours $\delta(etat,\ symbole)$ sera traduite par $M[i,\ j]$ où $i = etat$ et $j = indice(symbole)$.

La fonction "est_final(etat e)" permet de tester si un état e est final ou non. La fonction "accpter( )" simule la reconnaissance d'un mot par un **AFD**. On supposera que les mots tapés seront des mots sur l'alphabet A spécifié. Voici le squelette général du programme :

```
#include <stdio.h>

#define N  ...
#define K  ...
#define NF ...

char A[K] = {...};   /* alphabet */
int  M[N][K] = {{...}, ..., {...}};
int F[NF] = {...};

int est_final(int etat)
{
   int i;

   for(i = 0; i < NF; i++)
     if(F[i] == etat)
       return 1;
   return 0;
}

int indice(char symbole)
{
   int i;

   for(i = 0; i < K; i++)
     if(A[i] == symbole)
       return i;
}

int accepter()
{
```

```
   int car;
   int etat;

   etat = 0; /* On part de l'état initial */
   do
   {
      car = getchar();
      if(car != '\n')
      {
         etat = M[etat][indice(car)];
      }
   }while(car != '\n');
   return est_final(etat);
}

int main()
{
   printf("Votre mot : ");
   if(accepter())
     printf("Accepte\n");
   else
     printf("Non accepte\n");
   system("pause");
   return 0 ;
}
```

Ce programme est général pour n'importe quel automate : il faut tout simplment changer les données sur l'**AFD** considéré (remplir les trois points ...).
Voici maintenant les quatre **AFD** demandés :

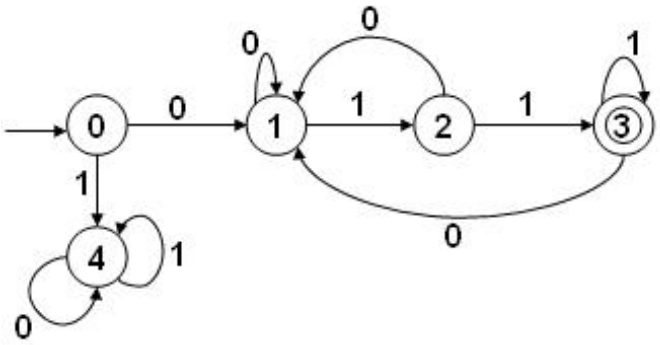

FIG. 1 – Un **AFD** complet de la question 1, Exercice 1.

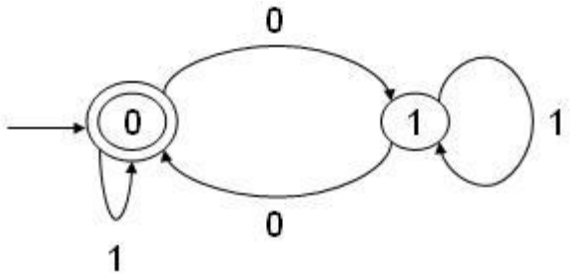

FIG. 2 – Un **AFD** complet de la question 2, Exercice 1.

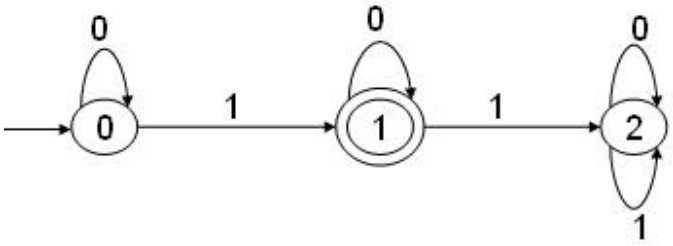

FIG. 3 – Un **AFD** complet de la question 3, Exercice 1.

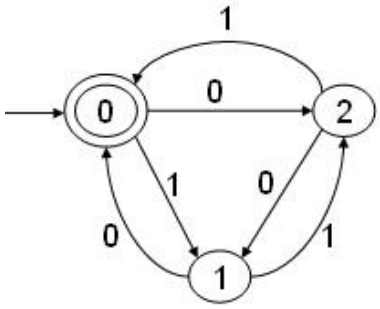

FIG. 4 – Un **AFD** complet de la question 4, Exercice 1.