DEVOIR MAISON 2

Automates (implémentation en C)

Objectifs :

L'objectif de ce travail est d'implémenter en langage C un automate déterministe (DFA) qui vérifie que la date/heure rentrée est valide.

Vous allez réutiliser les notions de grammaire, d'états et de transitions utilisés précédemment.

L'utilisation des tableaux dynamiques ou des listes chaînées est obligatoire pour la représentation de l'information dans le programme.

Suivez l'exercice pas à pas avec les ressources qui sont fournies. Chaque étape est détaillée et les actions à faire sont marquées par « // TODO » dans les fichiers sources.

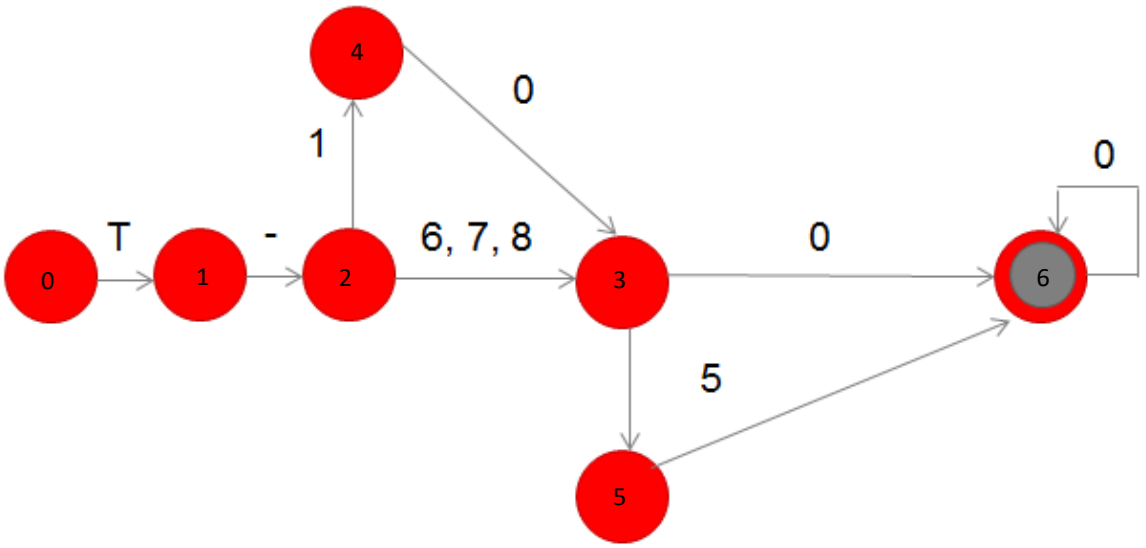
DEVOIR MAISON 2

Automates (implémentation en C)

1. AIDE A LA REALISATION

Vous avez dû construire dans le 1er DM sur les automates, un automate déterministe (DFA) qui vérifie que la date/heure rentrée est valide. Pour cela vous avez créé la grammaire, définit des états et des transitions.

Par exemple, voici l'automate suivant :



La table de transition de cet automate est une liste qui peut être représentée de la façon suivante :

Etat initial	Valeur lue	Etat final
0	T	1
1	-	2
2	1	4
2	6	3
2	7	3
2	8	3
3	0	6
4	0	3
3	5	5
5	0	6
6	0	6

Cette table d'état-transition représente l'automate.

2. EXERCICE OBLIGATOIRE : RECONNAISSANCE DE L'HEURE AU FORMAT HH:MM:SS

L'objectif est de réaliser, à partir de l'automate modélisé dans le 1er DM « modélisation d'automate », un programme en C qui reconnaît les heures au format : **hh:mm:ss** mêmes contraintes que dans l'énoncé de la modélisation des automates, pour rappel :

- la reconnaissance des minutes et secondes (mm:ss) - composées de 2 chiffres de 00 à 59.
- la reconnaissance des heures (hh:mm:ss) - les heures seront sur 2 chiffres de 00 à 23

1. ETAPE 1 : PRENDRE CONNAISSANCE DES RESSOURCES

- Ouvrir dans CodeBlock le projet « automates » que vous trouverez dans les ressources.
- Vous trouverez 3 fichiers .c et 2 fichiers .h.
- Qu'est-ce que les fichiers .h contiennent? Lisez les commentaires et essayez de décrire les structures et les fonctions avec vos propres mots.
- Dans les ressources vous trouverez aussi le fichier automate.txt qui contient la description d'un automate qui reconnaît l'heure selon le format indiqué ci-dessus. Voici les 4 premières lignes.

```
#0
#9
0;0;1
0;1;1
…
```

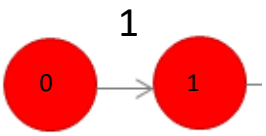
A la lumière de la table de transition du point 2 de ce WS, essayez de comprendre la logique de 2 premières lignes puis de toutes les autres.

EXPLICATIONS :

La première ligne donne le numéro de l'état initial de l'automate (après le #).
La deuxième ligne donne le numéro de l'état final de l'automate (après le #).
Chaque ligne restante du fichier représente une transition de l'automate (comme dans une table de transition). Le premier numéro est l'état initial. Le ';' est le séparateur entre 2 informations. Le troisième numéro est l'état final. Le numéro du

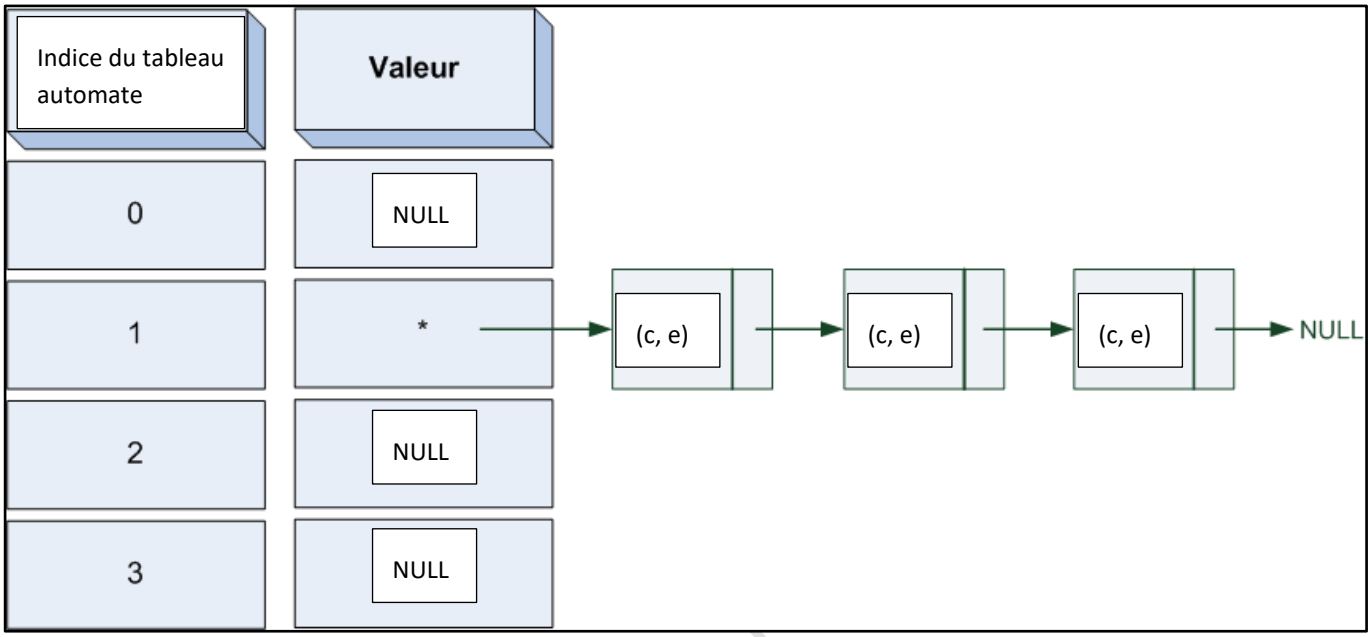
milieu est le « caractère » qui fait passer l'automate de l'état initial à l'état final (ou état suivant).

Voici la même chose représentée graphiquement pour la transition 0 ;1 ;1



A partir du fichier automate.txt, représentez l'automate graphiquement en lisant le fichier.

2. **ETAPE 2 :** COMPRENDRE LA STRUCTURE DES DONNEES



- L'indice du tableau 'automate' représente l'état initial d'une transition d'état.
- La valeur du tableau pointe vers la tête d'une liste chaînée où chaque maillon de la liste contient le couple (c,e). « c » représente le caractère de la transition et « e » l'état suivant.
- Le champ « next » de la structure est le pointeur vers la transition suivante. Toutes les transitions de la liste ont comme état initial l'indice du tableau 'automate'. Par exemple, la liste chainée d'indice 1 contient toutes les transitions partant de l'état 1.
- Si un état n'a pas de transitions, il pointe vers NULL.

- A vous d'expliquer et représenter la variable « char *tableHeuresTests[NBTEST]; »

3. **ETAPE 3 :** FICHIER ENTREESORTIE.C

Ouvrir le fichier « entreeSortie.c ».

A partir de la définition des fonctions, suivez les indications marquées « TODO » pour compléter les 2 fonctions.

En l'état le code n'est pas testable, mais vous pouvez compiler le code pour en vérifier la syntaxe et les erreurs de codage.

4. **ETAPE 4 :** FICHIER MAIN.C

Ouvrir le fichier « main.c ».

Dans cette étape, le travail consiste à compléter la fonction main selon les indications dans le fichier 'main.c'.

Pour réussir cette partie, vous devez bien regarder les variables locales et faire correspondre les variables locales du 'main' avec les paramètres des fonctions.

Pour l'initialisation du tableau 'automate', vous devez utiliser des boucles.

Pour les autres TODO du 'main', soit vous devez faire appel à des fonctions déjà définies dans les fichiers .h et passer les bons paramètres (valeur, adresse, pointeur, …), soit vous devez compléter avec une instruction en C.

La difficulté, peut-être perçue, est de remplir la fonction 'main' sans connaître l'implémentation des fonctions que vous utilisez. Cet exercice est très intéressant car il vous permet de vous concentrer sur l'appel des fonctions et le passage des paramètres aux fonctions sans forcément connaître leur implémentation.

Compilez à nouveau votre code. Désormais vous pouvez tester votre code. Pour que le programme fonctionne, vous devez avoir le fichier automate.txt et HeuresTests.txt dans le même répertoire que le .exe (ou paramétrer votre IDE pour qu'il le cherche dans le répertoire où se trouve votre projet)

5. ETAPE 5 : FICHIER ACTIONS.C

Ouvrir le fichier « Actions.c »

Complétez les fonctions dans cet ordre :

- **lireFichier** – essayez de comprendre les fonctions de lecture du fichier puis complétez les TODO. Ne réinventez pas la roue ! **Certaines lignes sont simplement des appels à des fonctions qui sont dans le même fichier 'actions.c'**. Pensez à passer les bons paramètres par valeur ou par variable.

- **ajouterTransition, creerTransition, libererTransitions** – complétez ces 3 petites fonctions. Pour bien compléter ces 3 fonctions, vous devez avoir compris avant la structure de données décrite dans l'étape 2.

- **chercherEtatSuivant** – complétez les TODO. Parcourir la liste chaînée tant que le caractère ne correspond pas au caractère passé en paramètre.

- **analyse** – complétez les TODO. Parcourir le « mot » lettre par lettre. Faire appel à la fonction « chercherEtatSuivant » pour chercher la transition suivante et tester si on a terminé la lecture du mot et si on se trouve dans l'état final de l'automate.

Compilez à nouveau votre code. Il devrait être fonctionnel. Si ce n'est pas le cas, n'hésitez pas à rechercher les bugs par déduction (c'est un très bon exercice), soit apprenez à utiliser le debugger de votre IDE.

3. EXERCICE OPTIONNEL : RECONNAISSANCE DES FORMATS PLUS COMPLEXES

Si vous êtes arrivé dans cette section c'est que vous avez réussi à faire le programme en C. Si tel est le cas, félicitations. Le plus gros du travail a été fait.

Maintenant vous pouvez, à partir des automates que vous avez réalisés pendant le DM sur la modélisation, écrire le fichier '*automate.txt*' correspondant **aux formats plus complexes (heure avec 1 ou 2 chiffres, date avec 12 mois de 31 jours, mois de 28,29,30 ou 31 jours, années bissextiles, notation française ou/et notation anglosaxone, …)**

Vérifiez que le même programme que vous avez réalisé dans l'étape précédente, est capable de lire ce fichier et de vérifier si un mot donné est une date/heure valide. Vous ne devriez pas adapter le code s'il fonctionne déjà pour le premier exercice.

Le fichier HeuresTests.txt peut servir pour les format d'heures plus complexe également.

Pour le test de dates, vous trouverez un autre fichier DatesTests.txt qui pour servira si vous arrivez à cette étape-là. Bon courage.