Corrigé TP10

1 Encore des automates ...

[a] Le code de la porte est **1234**. En effet, on peut voir qu'on arrive dans l'état initial `Q0` dans lequel tout chiffre différent de 1 ramène au même état. Puis, si on tape 1, on aboutit à l'état `Q1` dans lequel tout chiffre différent de 2 ramène à l'état `Q0`, et ainsi de suite jusqu'au chiffre 4. Seul la séquence d'entrée **1234** amène l'automate à produire la sortie **"clic"**.

[b] Pour arrêter la simulation, il faut entrer **"-"**. En effet, si on observe la fonction `lire_entree`, le programme demande à l'utilisateur d'entrer un caractère au clavier jusqu'à ce que le résultat soit un chiffre en 0 et 9, ou un tiret. Si c'est bien un chiffre qui est entré, la valeur du chiffre est renvoyée (`c - '0'`), sinon -1. Et la fonction `simule_automate` boucle tant que l'entrée est différente de -1.

[c] Je n'avais absolument pas remarquée la fonction pour tester si un état est final. On peut donc utiliser le retour de la fonction `est_final` dans la condition de notre `while` pour s'arrêter lorsque l'état final est atteint.

Voici une implémentation de Digicode_V1 (Seulement le fichier `automate.c`, les autres restent inchangés) :

```c
#include <stdio.h>
#include <string.h>
#include "automate.h"

#define NB_ETATS      5
#define NB_ENTREES    10
#define LG_MAX_SORTIE 128

#define Q0 0
#define Q1 1
#define Q2 2
#define Q3 3
#define Q4 4

char essai[]="/*" /* un essai : " */;
typedef int etat;

etat etat_initial() {
   return Q0;
}

int est_final(etat Q) {
   return Q == Q4;
}

int lire_entree() {
   char c;

   scanf("\n%c", &c);
   while (((c < '0') || (c > '9')) && (c != '-')) {
      printf("entree invalide ! un chiffre entre 0 et 9 (- pour finir) ?\n");
      scanf("\n%c", &c);
   }

   if (c == '-') {
      return -1;
   } else {
      return c-'0';
   }
}

void simule_automate() {
```

```
    int etat_courant, etat_suivant, entree;
    entree = 0;
    etat_courant = etat_initial();

    while (entree != -1 && !est_final(etat_courant)) {
        entree = lire_entree();

        switch (etat_courant) {
        case Q0:
            switch (entree) {
            case 1:  etat_suivant = Q1; break;
            default: etat_suivant = Q0; break;
            }
            break;

        case Q1:
            switch (entree) {
            case 1:  etat_suivant = Q1; break;
            case 2:  etat_suivant = Q2; break;
            default: etat_suivant = Q0; break;
            }
            break;

        case Q2:
            switch (entree) {
            case 1:  etat_suivant = Q1; break;
            case 3:  etat_suivant = Q3; break;
            default: etat_suivant = Q0; break;
            }
            break;

        case Q3:
            switch (entree) {
            case 1:  etat_suivant = Q1; break;
            case 4:
                etat_suivant = Q4;
                printf ("Clic !\n");
                break;
            default: etat_suivant = Q0; break;
            }
            break;

        case Q4:
            switch (entree) {
            case 1:  etat_suivant = Q1; break;
            default: etat_suivant = Q0; break;
            }
            break;

        default : break;
        }

        /* Trace de débug, à décommenter seulement si ça plante */
        /* printf("%d - %d -> %d\n", etat_courant, entree, etat_suivant); */

        etat_courant = etat_suivant;
    }
}
```

Voilà une implémentation possible pour la fonction `transition` et l'adaptation de la fonction `simule_automate` correspondante.

On note que la sortie **"clic"** est affichée dans la fonction de transition. Ce n'est pas complètement satisfaisant et l'on pourrait plutôt séparer cette sortie dans une fonction à part.

```
etat transition (etat Q, int e) {
   etat etat_suivant = Q;

   switch (Q) {
   case Q0:
      switch (e) {
      case 1:  etat_suivant = Q1; break;
      default: etat_suivant = Q0; break;
      }
      break;

   case Q1:
      switch (e) {
      case 1:  etat_suivant = Q1; break;
      case 2:  etat_suivant = Q2; break;
      default: etat_suivant = Q0; break;
      }
      break;

   case Q2:
      switch (e) {
      case 1:  etat_suivant = Q1; break;
      case 3:  etat_suivant = Q3; break;
      default: etat_suivant = Q0; break;
      }
      break;

   case Q3:
      switch (e) {
      case 1:  etat_suivant = Q1; break;
      case 4:
         etat_suivant = Q4;
         printf ("Clic !\n");
         break;
      default: etat_suivant = Q0; break;
      }
      break;

   case Q4:
      switch (e) {
      case 1:  etat_suivant = Q1; break;
      default: etat_suivant = Q0; break;
      }
      break;

   default : break;
   }

   return etat_suivant;
}

void simule_automate() {
   int etat_courant, etat_suivant, entree;
   entree = 0;
   etat_courant = etat_initial();

   while (entree != -1 && !est_final(etat_courant)) {
      entree = lire_entree();
      etat_suivant = transition (etat_courant, entree);

      /* Trace de débug, à décommenter seulement si ça plante */
      /* printf("%d - %d -> %d\n", etat_courant, entree, etat_suivant); */

      etat_courant = etat_suivant;
   }
}
```

[d] Voilà la représentation de la fonction de transition de notre digicode.

	1	2	3	4	0, 5, 6, 7, 8, 9
Q0	Q1	Q0	Q0	Q0	Q0
Q1	Q1	Q2	Q0	Q0	Q0
Q2	Q1	Q0	Q3	Q0	Q0
Q3	Q1	Q0	Q0	Q4	Q0
Q4	Q1	Q0	Q0	Q0	Q0

On note que dans cet représentation, les entrées 0, 5, 6, 7, 8, 9 ont été factorisées car l'automate se comporte de la même manière dans tous ces cas. Mais ce ne sera pas toujours le cas, en particulier si l'on change le code! Dans notre implémentation, nous choisirons donc de coder le tableau de transition en donnant une colone par chiffre.

Voilà donc une implémentation du tableau des transitions dans Digicode_V3, suivie de la fonction `simule_automate` adaptée :

```
etat transition [NB_ETATS][NB_ENTREES] = {
   /* entrees ->  0,  1,  2,  3,  4,  5,  6,  7,  8,  9 */
   /* Q0 */      { Q0, Q1, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
   /* Q1 */      { Q0, Q1, Q2, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
   /* Q2 */      { Q0, Q1, Q0, Q3, Q0, Q0, Q0, Q0, Q0, Q0 },
   /* Q3 */      { Q0, Q1, Q0, Q0, Q4, Q0, Q0, Q0, Q0, Q0 },
   /* Q4 */      { Q0, Q1, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 }
};

void simule_automate() {
   int etat_courant, etat_suivant, entree;
   entree = 0;
   etat_courant = etat_initial();

   while (entree != -1 && !est_final(etat_courant)) {
      entree = lire_entree();
      etat_suivant = transition[etat_courant][entree];

      if (etat_courant == Q3 && entree == 4) {
         printf ("Clic !\n");
      }

      /* Trace de débug, à décommenter seulement si ça plante */
      /* printf("%d - %d -> %d\n", etat_courant, entree, etat_suivant); */

      etat_courant = etat_suivant;
   }
}
```

[e] Pour la sortie, on ne peut évidemment plus inclure la seule sortie **"clic"** dans le tableau de transition, comme on le faisait dans l'ammas de switchs. On a donc pris le parti, dans cette implémentation, d'intégrer la sortie dans la fonction `simule_automate` à l'aide d'un petit `if`. On note bien que si l'automate avait plus sorties, il aurait sans doute fallu les traiter dans une fonction dédiée.

[f] Nous voilà donc arrivé au moment fatidique tant attendu : il faut maintenant changer le code de la porte. Dans la version Digicode_V3, le code est intégré dans le tableau de transition. Voilà un tableau de transition pour le code **2345** :

```
    etat transition [NB_ETATS][NB_ENTREES] = {
      /* entrees ->  0,  1,  2,  3,  4,  5,  6,  7,  8,  9 */
      /* Q0 */      { Q0, Q0, Q1, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
      /* Q1 */      { Q0, Q0, Q1, Q2, Q0, Q0, Q0, Q0, Q0, Q0 },
      /* Q2 */      { Q0, Q0, Q1, Q0, Q3, Q0, Q0, Q0, Q0, Q0 },
      /* Q3 */      { Q0, Q0, Q1, Q0, Q0, Q4, Q0, Q0, Q0, Q0 },
      /* Q4 */      { Q0, Q0, Q1, Q0, Q0, Q0, Q0, Q0, Q0, Q0 }
    };
```

Une autre implémentation pour le code **3874** :

```
etat transition [NB_ETATS][NB_ENTREES] = {
  /* entrees ->  0,  1,  2,  3,  4,  5,  6,  7,  8,  9 */
  /* Q0 */      { Q0, Q0, Q0, Q1, Q0, Q0, Q0, Q0, Q0, Q0 },
  /* Q1 */      { Q0, Q0, Q0, Q1, Q0, Q0, Q0, Q0, Q2, Q0 },
  /* Q2 */      { Q0, Q0, Q0, Q1, Q0, Q0, Q0, Q3, Q0, Q0 },
  /* Q3 */      { Q0, Q0, Q0, Q1, Q4, Q0, Q0, Q0, Q0, Q0 },
  /* Q4 */      { Q0, Q0, Q0, Q1, Q0, Q0, Q0, Q0, Q0, Q0 }
};
```

Le code peut bien sûr comporter plusieurs fois le même chiffre. Exemple avec un implémentation pour le code **1212** :

```
etat transition [NB_ETATS][NB_ENTREES] = {
  /* entrees ->  0,  1,  2,  3,  4,  5,  6,  7,  8,  9 */
  /* Q0 */      { Q0, Q1, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
  /* Q1 */      { Q0, Q1, Q2, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
  /* Q2 */      { Q0, Q3, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
  /* Q3 */      { Q0, Q1, Q4, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
  /* Q4 */      { Q0, Q1, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 }
};
```

[g] On remarque qu'on a modifié seulement le Digicode_V3 dans cette correction. Pour modifier le code dans les version 1 et 2, il aurait fallu modifier les switchs, ce qui est très fastidieux. Dans la version **tabulaire**, on peut se concentrer sur la logique de l'automate indépendemment de l'implémentation. Cela en fait une version beaucoup plus maintenable, c'est à dire facile à modifier en limitant le risque d'introduction de nouveaux bugs.

Bien entendu, quelle que soit la version choisie, seuls des tests rigoureux, et si possible automatiques, pourront nous donner confiance dans notre code.

[h] Dans les exemples précédents de table de transition, les états différents de `Q0` sont coloriés en rouge. On voit donc bien que taper le premier chiffre du code doit toujours mener à l'état `Q1`, quel que soit l'état de départ. En effet, taper le bon premier chiffre est toujours potentiellement le début d'une bonne séquence.

En revanche, taper les chiffres suivants du code ne doit mener à l'état correspondant que si on a entré précédemment les bons chiffres. Ensuite, taper n'importe quel chiffre qui n'est pas dans la combinaison ou qui est précédé d'une combinaison erronnée doit ramener à l'état d'origine `Q0`.

Voilà pourquoi la table contient partout des `Q0`, sauf sur la colonne correspondant au premier chiffre du code, où elle contient des `Q1`, puis seulement un `Q2`, un `Q3` et un `Q4` aux transitions correspondant à la bonne séquence de code entrée.

[i] Dans la suite, on lit l'automate depuis un fichier de sauvegarde. Avant de se plonger dans le listing, voilà quelques explications sur les choix techniques.

— Le format choisi pour le fichier est le suivant :

```
1
2
3
4
Porte ouverte !
```

On reconnait les quatre chiffres du code ainsi que le message qui s'affiche lorsqu'on a entré la bonne combinaison. Certains d'entre-vous auront peut être ré-employé la methode du TP9, en sauvegardant l'intégralité de la table de transition et de la table de sortie. Ce n'est absolument pas nécessaire dans ce cas. En effet, connaissant le code, et le message de déverrouillage désiré, nous pourrons reconstituer ces deux tables. Ce fichier de sauvegarde contient donc l'intégratlité des données dont nous avons besoin.

— Par ailleurs, on a placé chaque chiffre sur une ligne pour éviter de lire un seul int comme **1234**, ce qui aurait nécessité quelques caluls pour extraire les chiffres individuellements. On a le choix du format, alors on en profite !

— Dans le fichier `automate.c`, vous ne trouverez pas la fonction `init_par_defaut` demandée dans le sujet du TP. C'est parce qu'elle est redondante avec l'initialisation de mon tableau de transition tel que je l'ai réalisé.

— On ajoute donc deux fonctions, `lecture_automate` et `init_automate` qui vont respectivement lire le fichier de sauvegarde et configurer la table de transition avec les valeurs obtenues. La fonction `lecture_automate` devra maintenant être appelée dans le `main` avant d'utiliser l'automate. Donc elle doit être visible depuis l'extérieur du fichier `automate.c` ce qui justifie qu'elle soit déclarée dans le fichier `automate.h`

— Observée l'élégance avec laquelle j'ai employé une structure pour stocker le message de sortie !

Sans plus attendre voici le code de `automate.c` :

```c
#include <stdio.h>
#include <string.h>
#include "automate.h"

#define NB_ETATS      5
#define NB_ENTREES    10
#define LG_MAX_SORTIE 128

#define Q0 0
#define Q1 1
#define Q2 2
#define Q3 3
#define Q4 4

typedef int etat;

typedef struct sortie_deverouillage {
   etat etat;
   int entree;
   char message[LG_MAX_SORTIE];
} Sortie_Deverouillage;

etat transition [NB_ETATS][NB_ENTREES] = {
   /* entrees ->  0,  1,  2,  3,  4,  5,  6,  7,  8,  9 */
   /* Q0 */     { Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
   /* Q1 */     { Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
   /* Q2 */     { Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
   /* Q3 */     { Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 },
   /* Q4 */     { Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0, Q0 }
};

Sortie_Deverouillage sortie;

int init_automate (int digit1, int digit2, int digit3, int digit4, char *message) {
   if (digit1 < 0 || digit1 > 9 || digit2 < 0 || digit2 > 9 ||
```

```
        digit3 < 0 || digit3 > 9 || digit4 < 0 || digit4 > 9) {
        return 1;
    }

    transition[Q0][digit1] = Q1;
    transition[Q1][digit1] = Q1;
    transition[Q2][digit1] = Q1;
    transition[Q3][digit1] = Q1;
    transition[Q4][digit1] = Q1;

    transition[Q1][digit2] = Q2;
    transition[Q2][digit3] = Q3;
    transition[Q3][digit4] = Q4;

    strcpy (sortie.message, message);
    sortie.etat = Q3;
    sortie.entree = digit4;

    return 0;
}

int lecture_automate (char *sauvegarde) {
    FILE *f = fopen (sauvegarde, "r");
    if (f == NULL) {
        return 1;
    }

    int digit1;
    fscanf (f, "%d\n", &digit1);

    int digit2;
    fscanf (f, "%d\n", &digit2);

    int digit3;
    fscanf (f, "%d\n", &digit3);

    int digit4;
    fscanf (f, "%d\n", &digit4);

    char message[LG_MAX_SORTIE];
    int i=0;
    char cc;
    fscanf (f, "%c", &cc);
    while (!feof(f) && cc != '\n') {
        message[i] = cc;
        fscanf (f, "%c", &cc);
        i++;
    }
    message[i] = '\0';

    fclose (f);
    return init_automate (digit1, digit2, digit3, digit4, message);
}

etat etat_initial() {
    return Q0;
}

int est_final(etat Q) {
    return Q == Q4;
}

int lire_entree() {
    char c;

    scanf("\n%c", &c);
    while (((c < '0') || (c > '9')) && (c != '-')) {
        printf("entree invalide ! un chiffre entre 0 et 9 (- pour finir) ?\n");
        scanf("\n%c", &c);
    }

    if (c == '-') {
        return -1;
    } else {
```

```
        return c-'0';
    }
}

void simule_automate() {
    int etat_courant, etat_suivant, entree;
    entree = 0;
    etat_courant = etat_initial();

    while (entree != -1 && !est_final(etat_courant)) {
        entree = lire_entree();
        etat_suivant = transition[etat_courant][entree];

        if (etat_courant == sortie.etat && entree == sortie.entree) {
            printf ("%s\n", sortie.message);
        }

        /* Trace de débug, à décommenter seulement si ça plante */
        /* printf("%d - %d -> %d\n", etat_courant, entree, etat_suivant); */

        etat_courant = etat_suivant;
    }
}
```

Puis le code de `automate.h` :

```
#ifndef __AUTOMATE__
#define __AUTOMATE__

void simule_automate();
int lecture_automate(char *);

#endif
```

Et enfin le code de `main.c`

```
#include <stdio.h>
#include <stdlib.h>
#include "automate.h"


int main() {
    if (lecture_automate("sauvegarde.sav")) {
        printf ("Impossible d'initialiser l'automate\n");
        exit(1);
    }

    simule_automate();
    return 0;
}
```

Sans oublier le fichier de sauvegarde :

```
2
3
2
5
Porte ouverte !
```