Exercice 8 : Machines de Turing

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <array>
using namespace std;

// ----------------------------------------------------------------------
// STRUCTURES DE DONNEES
// ----------------------------------------------------------------------
typedef string Bande;
constexpr char EPSILON('e');

typedef int Tete;
constexpr Tete POSITION_INITIALE(0);

typedef unsigned int Etat;
constexpr Etat UNDEF(0);
constexpr Etat ETAT_INITIAL(1);

struct Transition
{
  Etat etat        ;
  char caractere   ;
  int  deplacement ;
};

typedef array<Transition, 3> Ligne; // [ 0 , 1 , epsilon ]

typedef vector<Ligne> TableTransition;

struct TMachine {
  Etat  actuel;
  Bande ruban ;
  Tete  tete  ;
  TableTransition table;
};

// ----------------------------------------------------------------------
// PROTOYPES
// ----------------------------------------------------------------------
char lecture(Bande& b, Tete& t); // lit le caractère 'courant'
void ecriture(char c, Bande& b, Tete& t); /* remplace le caractère courant par
        * la valeur transmise en argument */
void avance(Tete& t) {++t;} // déplace la tête de lecture vers la droite
void recule(Tete& t) {--t;} // déplace la tête de lecture vers la gauche
void initialise(Bande& b, const string& valeur) {b=valeur;}
TMachine cree(const string& entree = "");
void reinitialise(TMachine& m, const string& entree);
void lance(TMachine& m);
void lance(TMachine&& m) { lance(m); }
void affiche(const TMachine& m);

// ----------------------------------------------------------------------
int main()
{
  lance(cree("1101"));
  return 0;
}

/* ========================================================================= *
 * Simule l'aspect infini négatif en ajoutant un caractère en debut de     *
 * chaîne puis en décalant la valeur de la tête (càd la remettre à 0).     *
 * Entrée : la bande et la tête de lecture/écriture                        *
 * ========================================================================= */
void gestion_bornes(Bande& b, Tete& t)
```

```
{
  // simulation du ruban pour les positions négatives
  while (t < 0) {
    b = EPSILON + b;
    avance(t);
  }
  // insertion des EPSILON nécessaires pour les position positives
  while (t >= b.size()) { // ici t >= 0 : comparaison OK
    b += EPSILON;
  }
}

/* ========================================================================== *
 * Lit le caractère courant sur une bande doublement infinie.                 *
 * Entrée : la bande à lire et la tête de lecture/écriture                    *
 * Sortie : le caractère lu                                                   *
 * ========================================================================== */
char lecture(Bande& b, Tete& t)
{
  // gère les bornes de la bande pour assurer l'existence de la case lue
  gestion_bornes(b, t);

  // lecture du caractère
  return b[t];
}

/* ========================================================================== *
 * Écriture d'un caractère sur une bande doublement infinie :                 *
 * remplace le caractère courant par la valeur transmise en argument.        *
 * Entrée : le caractère à écrire, la bande et la tête de lecture/écriture*
 * ========================================================================== */
void ecriture(char c, Bande& b, Tete& t)
{
  // gère les bornes de la bande pour assurer l'existence de la case écrite
  gestion_bornes(b, t);

  // écriture du caractère
  b[t] = c;
}

/* ========================================================================== *
 * Crée une machine de Turing : charge sa table de transition et              *
 * initialise la machine.                                                     *
 * ========================================================================== */
TMachine cree(const string& entree)
{
  TMachine m;
   m.table = {
    // table pour le calcul de parité (g++-4.6)
     {{ {    1,     '0',  1}, {    1,     '1',  1}, {    2, EPSILON, -1} }},
     {{ {    3, EPSILON, -1}, {    4, EPSILON, -1}, {UNDEF, EPSILON, -1} }},
     {{ {    3, EPSILON, -1}, {    3, EPSILON, -1}, {    5,     '1',  1} }},
     {{ {    4, EPSILON, -1}, {    4, EPSILON, -1}, {    5,     '0',  1} }},
     {{ {UNDEF, EPSILON, -1}, {UNDEF, EPSILON, -1}, {UNDEF, EPSILON, -1} }}

    /*
    // table pour le calcul de parité (normalement)
    { {    1,     '0',  1}, {    1,     '1',  1}, {    2, EPSILON, -1} },
    { {    3, EPSILON, -1}, {    4, EPSILON, -1}, {UNDEF, EPSILON, -1} },
    { {    3, EPSILON, -1}, {    3, EPSILON, -1}, {    5,     '1',  1} },
    { {    4, EPSILON, -1}, {    4, EPSILON, -1}, {    5,     '0',  1} },
    { {UNDEF, EPSILON, -1}, {UNDEF, EPSILON, -1}, {UNDEF, EPSILON, -1} }
    */

    /*
    // table pour l'inversion de bits
    { {    2,     '0',  1}, {    3,     '1',  1}, {UNDEF, EPSILON,  1} },
    { {    2,     '0',  1}, {    3,     '1',  1}, {    4, EPSILON,  1} },
    { {    2,     '0',  1}, {    3,     '1',  1}, {    5, EPSILON,  1} },
```

```
    { {    4,     '0',  1}, {    4,     '1',  1}, {    6,     '0', -1} },
    { {    5,     '0',  1}, {    5,     '1',  1}, {    6,     '1', -1} },
    { {    6,     '0', -1}, {    6,     '1', -1}, {    7, EPSILON, -1} },
    { {    8, EPSILON, -1}, {    8, EPSILON, -1}, {    7, EPSILON, -1} },
    { {   10,     '0',  1}, {   11,     '1',  1}, {    9, EPSILON,  1} },
    { {UNDEF,     '0', -1}, {UNDEF,     '1', -1}, {    9, EPSILON,  1} },
    { {    4,     '0',  1}, {    4,     '1',  1}, {   10, EPSILON,  1} },
    { {    5,     '0',  1}, {    5,     '1',  1}, {   11, EPSILON,  1} }
    */

  };

  reinitialise(m, entree);

  return m;
}

/* ================================================================== *
 * Réinitialise une machine et écrit les données d'entrée sur la bande.   *
 * Entrée : la machine à initialiser et les données d'entrée          *
 * ================================================================== */
void reinitialise(TMachine& m, const string& entree)
{
  m.actuel = ETAT_INITIAL;
  initialise(m.ruban, entree);
  m.tete = POSITION_INITIALE;
}

/* ================================================================== *
 * Lance l'execution d'une machine de Turing                          *
 * Entrée : la machine à simuler                                      *
 * ================================================================== */
void lance(TMachine& m)
{
  cout << "Lancement de la machine de Turing" << endl;
  affiche(m);

  while (m.actuel != UNDEF) {
    // Cherche la position à lire dans la table de transition

    //      indice de ligne
    int i(m.actuel-1);

    //      indice de colonne (caractère lu)
    cout << " lu : " << lecture(m.ruban, m.tete) << ",";
    int j;
    switch (lecture(m.ruban, m.tete)) {
    case '0':     j = 0; break;
    case '1':     j = 1; break;
    case EPSILON: j = 2; break;
    default:
      cout << "ERREUR: j'ai lu un caractère inconnu: "
    << lecture(m.ruban, m.tete) << endl;
      affiche(m);
      return;
    };

    // mise à jour de la machine
    m.actuel = m.table[i][j].etat;
    cout << " nouvel état : " << m.actuel << ",";

    ecriture(m.table[i][j].caractere, m.ruban, m.tete);
    cout << " écrit : " << m.table[i][j].caractere << ",";

    switch(m.table[i][j].deplacement) {
    case  1: avance(m.tete); cout << " avance" << endl; break;
    case -1: recule(m.tete); cout << " recule" << endl; break;
    default:
      cout << "ERREUR: déplacement inconnu: "
```

```
    << m.table[i][j].deplacement << endl;
      affiche(m);
      return;
    }

    affiche(m);
  }

  cout << "Arrêt de la machine de Turing." << endl;
}

/* ================================================================== *
 * Affiche l'état d'une machine de Turing                             *
 * Entrée : la machine à afficher                                     *
 * ================================================================== */
void affiche(const TMachine& m)
{
  cout << "    ---------------------------------" << endl;
  cout << "    Etat actuel : " << m.actuel << endl;
  cout << "    Bande :" << endl;
  cout << "       " << m.ruban << endl;
  cout << "      ";
  for (Tete i(-1); i < m.tete; ++i) cout << ' ';
  cout << "^ (Tete : " << m.tete << ')' <<  endl;
  cout << "    ---------------------------------" << endl;
}
```