Machine de Turing (programmation, niveau 3)

> Cet exercice n'est pas présent dans l'ouvrage *C++ par la pratique*

Le but est d'écrire un programme `turing.cc` permettant de simuler une machine de Turing.

Si vous n'avez aucune idée sur la manière de procéder, vous pouvez utiliser la modélisation **proposée** ci-dessous, sinon allez y puis **reprenez l'exercice ici**.

Proposition de méthode :

1. Pour simuler une bande de longueur infinie, vous pouvez utiliser une string (définir par exemple un type Bande) :
   - lorsque l'on cherchera à déplacer la tête au-delà du dernier élément, il suffira de créer un élément supplémentaire, initialisé avec le symbole 'epsilon' (le caractère correspondant, à définir).
   - lorsque l'on cherchera à déplacer la tête avant le premier élément, il suffira d'insérer en début de chaîne le symbole 'epsilon'.
2. La tête de lecture sera représentée simplement par un entier, indiquant sa position courante. Je vous conseille aussi de définir un type ici, par exemple Tete.
3. Créez ensuite les fonctions implémentant les primitives de lecture, écriture et positionnement de la tête de lecture :
   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); *déplace la tête de lecture vers la droite*
   void recule(Tete& t); *déplace la tête de lecture vers la gauche*

   Notez que, dans l'implémentation proposée, lecture et ecriture peuvent changer la valeur de tete : comme indiqué au point 1. ci-dessus, l'aspect infini de la bande est simulé en insérant le bon nombre d'epsilons en début de bande lorsque la position de la tête devient négative. Il faut alors naturellement repositionner la tête à la valeur zéro lorsque cela a été fait, et donc modifier sa valeur. On aurait également pu choisir d'effectuer ces opérations (simuler une bande infinie) lors des déplacements de la tête plutôt que lors de la lecture/écriture... (libre à vous)

4. Définissez une fonction initialise(Bande& b, string valeur) permettant d'initialiser la bande au moyen d'une chaîne de caractères
5. Pour représenter la table de transitions, utilisez un tableau dynamique :
   - chaque 'case' définissant une transition est une structure de 3 champs ([prochain état, caractère, déplacement]);
   - chaque ligne de transition est un tableau de 3 'cases', une par valeur possible du caractère courant ([0, 1, epsilon]);

   ```
   struct Transition
   {
     Etat etat;
     char caractere;
     int deplacement;
   }
   typedef array Ligne; // 3 : caractères  0 , 1 , epsilon
   ```

   Avec une telle représentation, une table de transitions sera simplement un tableau de Lignes.

6. Écrivez une structure **TMachine**, simulant une machine de Turing.

   Vous aurez besoin des champs suivants :

   - un entier (ou un type prédéfini par vous) modélisant l'**état actuel** de la machine;
   - une bande ;
   - une tête de lecture
   - une table de transitions
7. Définissez les fonctions suivantes :
   - une fonction permettant de créer une machine de Turing
     void cree(TMachine& m, TableTransition const& table);
   - une fonction permettant de ré-initialiser la machine (pour recommencer avec une nouvelle entrée)
     void reinitialise(TMachine& m, string_view entree);
   - une fonction permettant de démarrer la simulation de la machine.
     Choisissez une convention sur la valeur de l'état pour stopper la machine (par exemple, une valeur négative).

Application 1: Test de Parité

Testez votre machine avec l'exemple du cours testant la parité de l'entrée.

Avec la modélisation proposée, utilisez la syntaxe suivante pour définir une table de transition (dans la fonction cree :

```
TableTransition table = {
  { {    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} }
};
```

Note : il se peut qu'avec la version actuelle (4.6) du compilateur C++11 vous ayez à écrire :

```
TableTransition table = {
  {{ {    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} }}
};
```

là où l'écriture précédente aurait du fonctionner...

La constante UNDEF représente un état non défini également utilisé pour "l'ordre de fin d'exécution".

Dans cet exemple, on suppose que le déplacement vers l'avant est représenté par la valeur +1, et le déplacement vers l'arrière par -1.

Exemple (détaillé) d'execution :

```
Lancement de la machine de Turing
    --------------------------------
    Etat actuel : 1
    Bande :
       10
       ^ (Tete : 0)
    --------------------------------
 lu : 1, nouvel état : 1, écrit : 1, avance
    --------------------------------
    Etat actuel : 1
    Bande :
       10
        ^ (Tete : 1)
    --------------------------------
 lu : 0, nouvel état : 1, écrit : 0, avance
    --------------------------------
    Etat actuel : 1
    Bande :
       10
         ^ (Tete : 2)
    --------------------------------
 lu : e, nouvel état : 2, écrit : e, recule
    --------------------------------
    Etat actuel : 2
    Bande :
       10e
        ^ (Tete : 1)
    --------------------------------
 lu : 0, nouvel état : 3, écrit : e, recule
    --------------------------------
    Etat actuel : 3
    Bande :
       1ee
       ^ (Tete : 0)
    --------------------------------
 lu : 1, nouvel état : 3, écrit : e, recule
    --------------------------------
```

```
    Etat actuel : 3
    Bande :
       eee
      ^ (Tete : -1)
    --------------------------------
 lu : e, nouvel état : 5, écrit : 1, avance
    --------------------------------
    Etat actuel : 5
    Bande :
       1eee
        ^ (Tete : 1)
    --------------------------------
 lu : e, nouvel état : 0, écrit : e, recule
    --------------------------------
    Etat actuel : 0
    Bande :
       1eee
       ^ (Tete : 0)
    --------------------------------
Arrêt de la machine de Turing.
```

Application 2: Inversion de séquence sur une Machine de Turing

Écrivez (sur papier) la table de transition d'une machine de Turing réalisant l'inversion d'une séquence de bits.

Par exemple, si l'on a 1101 en entrée, on devra avoir 1011 en sortie.

On supposera que:

- la bande (de longueur infinie) ne contient rien d'autre que la séquence (i.e. à l'exception de la séquence, tous les caractères sont des 'epsilon' -> la séquence est encadrée [délimitée] par des 'epsilon');
- la tête de lecture/écriture est initialement positionnée sur le premier caractère de la séquence.

Remarquez bien qu'il n'est pas nécessaire que la séquence inversée occupe, sur la bande, la même position que la séquence initiale.

Testez votre machine sur un exemple simple (du moins dans un premier temps) !

Indications: si vous êtes en panne d'inspiration, vous pouvez essayer l'algorithme **proposé** ci-après.

1. Si la séquence d'entrée est vide, terminer. Sinon, aller jusqu'à la cellule immédiatement à droite de la frontière droite de la séquence, en mémorisant (dans la valeur des états : voir l'algorithme de parité ci-dessus) la valeur du dernier élément de la séquence.
2. Si la (nouvelle) séquence n'est pas vide, aller tout d'abord à sa frontière droite (toujours en mémorisant le dernier élément), puis écrire l'élément mémorisé (qui est alors le dernier élément de la nouvelle séquence), se déplacer à la fin de la séquence d'entrée courante, effacer son dernier élément et se déplacer à gauche. (Si la séquence n'est pas vide, on est alors de nouveau sur le dernier élément de la séquence d'entrée 'courante')
3. Si la séquence d'entrée courante est vide, aller à la frontière gauche de la nouvelle séquence et terminer. Sinon, aller au début de la nouvelle séquence en mémorisant le dernier caractère de la séquence d'entrée courante, puis recommencer en (2)