Phase 1 : mes premières fonctions

Dans cette première phase, on se contente de créer des automates de Thompson à partir d'expressions régulières postfixées. Le fichier `phase1.h` contient toutes les structures détaillées ci-dessous.

1 Structures

Les états d'un automate de Thompson pouvant avoir une transition sortante étiquettée par une lettre OU un ϵ-transition OU 2 ϵ-transitions OU être un état terminal, la structure pour un état donné va devoir s'adapter à 4 cas de figure, d'où l'utilisation d'une `union`.

On définit des constantes pour désigner chaque catégorie d'état :

```
enum typeEtat {TRANSITION, EPSILON1, EPSILON2, TERMINAL};
```

Et une structure pour les états proprement dits :

```
struct etatStruct {
  int numero;
  enum typeEtat type;
  union{
    struct transition { // si type = TRANSITION
      char lettre; struct etatStruct *etat;
    } transition;
    struct epsilon1 {   // si type = EPSILON1
      struct etatStruct *etat1;
    } epsilon1;
    struct epsilon2 {   // si type = EPSILON2
      struct etatStruct *etat1, *etat2;
    } epsilon2;
    // si type = TERMINAL on n'a besoin de rien
  } fonction;
};
```

Pour simplifier le code source, on définit aussi le type `etat` :

```
typedef struct etatStruct etat;
```

Un automate de Thompson est alors un ensemble d'états, dont 2 distingués : l'état initial et l'état final.

```
struct automateStruct {
  int nbEtats;  // nombre d'états.
  etat *etats;  // tableau des états
  etat *etatInitial;  // pointeur sur l'état initial
  etat *etatFinal;    // pointeur sur l'état final
};
```

Et pour simplifier le code source, le type `automate` est défini :

```
typedef struct automateStruct automate;
```

2 Fonctions

La première fonction à écrire est celle qui permet de visualiser un automate de Thompson, pour pouvoir ensuite contrôler les effets des fonctions suivantes :

```
void impAutomate(automate *autom, char *alphabet);
```

`autom` est un pointeur sur un automate de Thompson, et `alphabet` est une chaîne des caractères de l'alphabet utilisé.

En vue de l'utilisation d'expressions régulières, écrire une fonction :

```
int  checkExpression(char *express, char *alphabet);
```

retournant `-1` si l'expression passée dans `express` est incorrecte, et $2n$ sinon (où n est comme dans le cours, le nombre de caractères de `express` qui sont dans `alphabet`$\cup\{*,+\}$).

Ensuite, il faut (dans l'ordre) :

1. une fonction qui crée l'automate de Thompson reconnaissant un caractère `c` donné :

   ```
   automate* lettre(char c);
   ```

2. une fonction retournant un automate de Thompson qui reconnaît l'union de 2 automates de Thompson `a` et `b` :

   ```
   automate* somme(automate* a,automate* b);
   ```

   La méthode à suivre est celle du cours, et vous pouvez stocker les états des automates comme vous voulez (par exemple dans un grand tableau commun)

3. une fonction pour le produit :

   ```
   automate* produit(automate* a,automate* b);
   ```

4. une fonction pour l'étoile d'un automate `a` :

   ```
   automate* etoile(automate* a);
   ```

Enfin une fonction construisant l'automate de Thompson d'une expression :

```
automate *construireAutomate(char *expression, char *alphabet);
```

Par convention, si l'expression est incorrecte, la fonction doit retourner `NULL`.