PROJET - CONCEPTION ET IMPLEMENTATION D'UN COMPILATEUR D'UN LANGAGE DE DESCRIPTION D'AUTOMATES FINIS DETERMINISTE.

1. OBJECTIF

Dans ce projet, nous allons concevoir et implémenter ensemble un compilateur complet pour un langage spécialisé permettant de décrire des automates finis déterministes (AFD).
Ce travail va nous permettre de comprendre de manière pratique comment fonctionne un compilateur, tout en appliquant ces connaissances à un domaine important : les automates.
Les automates finis déterministes jouent un rôle essentiel en informatique théorique et dans de nombreuses applications : analyse lexicale, protocoles, reconnaissance de motifs, etc. Créer un langage qui permet de les décrire simplement est donc un excellent exercice pour combiner théorie et mise en pratique.
Au cours du projet, nous allons :

- Définir notre propre langage de description d'automates,
- Concevoir sa syntaxe et sa sémantique,
- Implémenter un analyseur lexical, un analyseur syntaxique,
- Effectuer les vérifications sémantiques,
- Construire l'automate correspondant sous forme de structure de données ou de code.

Ce projet nous permettra de :

- Comprendre les différentes étapes de compilation,
- Manipuler des outils comme Flex/Bison (ou équivalents),
- Appliquer des notions théoriques (AFD, grammaires, arbres syntaxiques),
- Travailler en équipe sur un projet structuré et motivant.

L'objectif final est qu'à la fin du projet, nous puissions écrire un automate en utilisant notre langage, et notre compilateur sera capable de le comprendre et de le construire correctement.

2. LE LANGAGE DE DESCRIPTION D'AUTOMATES FINIS DETERMINISTE.

Le langage de description d'automates finis déterministe va permettre á ses utilisateurs d'écrire un automate de manière simple, structurée et lisible. Voici ces principales caractéristiques :

1. Mots clés réservés :

Nous allons définir des mots clés spécifiques que le programmeur ne peut pas utiliser comme noms d'états :

- automate
- alphabet
- etats

- initial
- finaux
- transitions
- verifier

Ces mots-clés nous permettront de structurer correctement l'analyse syntaxique.

2. Les commentaires :

Nous allons permettre aux utilisateurs du langage A d'utiliser des commentaires sur une seule ligne en commençant par le caractère spécial #.

3. Les identifiants :

Les identifiants (noms d'états, nom de l'automate) suivront une règle simple :

- Commencent par une lettre.
- Suivie de lettres ou chiffres.
- Pas d'accents ni caractères spéciaux.

Exemples valides : q0, Etat1, A, S3

4. L'alphabet :

L'alphabet sera généralement une liste de symboles, chacun composé d'un caractère simple `{a, b, c…}`.

- Nous allons limiter les symboles aux lettres de l'alphabet minuscule.

Exemple : `alphabet = {a, b, c}`

5. Les états :

Les états seront une liste des identifiants représentant les états de l'automate.

Exemple : `etats = {q0, q1, q2}`

6. Les transitions :

Les transitions seront une liste des transitions sous la forme :

```
etat_source : symbole -> etat_destination;
```

Exemple :
```
transitions = {
    q0 : a -> q1;
    q1 : b -> q2;
    }
```

7. Validation d'un mot :

La vérification d'un mot par un automate va se faire par l'instruction :

```
verifier nomAutomate "mot" ;
```

8. Les délimiteurs :

Nous allons utiliser :

- `{ }` pour les blocs,
- `;` pour terminer les instructions,
- `=` pour l'affectation,
- `:` et `->` pour exprimer une transition.

Voici un exemple d'un programme en A :

```
automate NomAutomate { #déclaration de l'automate
    alphabet = {a, b, c}; #déclarations des alphabets
```

```
    etats = {q0, q1, q2}; #déclaration des états
    initial = q0; #spécification de l'état initial
    finaux = {q2}; #spécification des états finaux
    transitions = { #déclaration des transitions.
        q0 : a -> q1; #premiére transition
        q1 : b -> q2; #deuxieme transition
        q2 : c -> q0; #troisieme transition
    };
} #fin du bloc de déclaration de l'automate NomAutomate.
verifier NomAutomate "ab"; #vérification du mot « ab » par l'automate.
```

Notre compilateur devra vérifier des règles comme :

- Les symboles utilisés dans les transitions doivent appartenir à l'alphabet.
- Les états dans les transitions doivent exister dans la liste des états.
- Il doit exister un seul état initial.
- L'automate doit être déterministe : pas de deux transitions avec le même symbole depuis un même état.

C'est votre compilateur ; vous pouvez y ajouter ce que vous souhaitez par la suite.
Si vous avez des remarques, merci de bien vouloir nous les faire parvenir.