



Projet Compilation 2

Liste des sujets

- Après avoir choisi un sujet de la liste:
- Vous pouvez ajouter des fonctionnalités supplémentaires
- Optionnellement, vous pouvez introduire l'usage de l'IA pour les tâches qui ne peuvent pas, ou difficilement, être accomplies avec la théorie de la compilation
- Livrables à rendre:
 - La grammaire au format BNF (pas obligatoirement LL(1)).
 - La grammaire au format EBNF.
 - Le fichier de description de la grammaire (Par exemple, pour JavaCC le fichier .jj)
 - Une description des fonctionnalités de l'application.
 - Les captures d'écran de l'application.
 - Le dépôt github du projet sur github classroom

Sujet 1: Le langage source C1

- Exemple de programme

X:=350;

y:=X*3 - 25 / 5;

afficher (y+5);

- Caractéristiques du langage:
 - Un programme peut avoir 1 ou plusieurs instructions.
 - Le langage doit supporter deux types d'instruction: affectation et affichage
 - Une instruction doit se terminer par un point virgule
 - Toutes les variables ont le même type de données (double).
 - La valeur par défaut des variables est 0.

```

<biblio>
  <livre>
    <numero>123</numero>
    <titre>Dialogues Sur L'Epicurisme Et Le Stoicisme</titre>
    <auteur>Ciceron</auteur>
    <editeur>La Bibliotheque De L'Antiquite</editeur>
    <prix>60</prix>
  </livre>
  <livre>
    <numero>124</numero>
    <titre>Le Pavillon d'Or</titre>
    <auteur>Yukio Mishima</auteur>
    <editeur>Gallimard</editeur>
    <prix>45</prix>
  </livre>
  <livre>
    <numero>125</numero>
    <titre>Discours de la méthode</titre>
    <auteur>René Descartes</auteur>
    <editeur>Flamamrion</editeur>
    <prix>15</prix>
  </livre>
  <livre>
    <numero>126</numero>
    <titre>Nouveaux essais sur l'entendement humain</titre>
    <auteur>Gottfried Wilhelm Leibniz</auteur>
    <editeur>Flamamrion</editeur>
    <prix>15</prix>
  </livre>
  <livre>
    <numero>127</numero>
    <titre>Critique de la raison pure</titre>
    <auteur>Emmanuel Kant</auteur>
    <editeur>puf</editeur>
    <prix>35</prix>
  </livre>
</biblio>

```

Sujet 2: Transformation de données

On suppose que nous avons des données qui sont stockées dans des documents xml, et nous souhaitons écrire un parseur pour valider le formats de ces fichiers xml.

Si le fichier xml est correct le parseur doit convertir les données au format JSON, le résultat doit être similaire à celui de l'application l'application oxygenxml:

https://www.oxygenxml.com/xml_editor/xml_json_converter.html

XML

Parcourir... Aucun fichier sélectionné.

```
1 <biblio>
2   <livre>
3     <numero>123</numero>
4     <titre>Dialogues Sur L'Epicurisme Et Le Stoicisme</titre>
5     <auteur>Ciceron</auteur>
6     <editeur>La Bibliotheque De L'Antiquite</editeur>
7     <prix>60</prix>
8   </livre>
9   <livre>
10    <numero>124</numero>
11    <titre>Le Pavillon d'Or</titre>
12    <auteur>Yukio Mishima</auteur>
13    <editeur>
14    Gallimard</editeur>
15    <prix>45</prix>45 </livre>
```

Download Copy Clear

JSON

Parcourir... Aucun fichier sélectionné.

```
1 {
2   "biblio": {
3     "livre": [
4       {
5         "numero": "123",
6         "titre": "Dialogues Sur L'Epicurisme Et Le Stoicisme",
7         "auteur": "Ciceron",
8         "editeur": "La Bibliotheque De L'Antiquite",
9         "prix": "60"
10      },
11      {
12        "numero": "124",
13        "titre": "Le Pavillon d\u2019Or",
14        "auteur": "Yukio Mishima",
15        "editeur": "\n  Gallimard",
```

Download Copy Clear

→

←

Sujet 3: DSL modélisation d'une classe

Définir un langage qui permet de décrire une classe UML en utilisant le format XML, chaque attribut de la classe doit avoir un type de données (les types de données doivent être limités, exemple: string, int, double) et une visibilité (privée, publique, protégée, package)

```
<Projet>
  <idProjet type="long" visibility="private"/>
  <description type="String" visibility="private"/>
  <date type="Date" visibility="private"/>
</Projet>
```

Projet		
-	idProjet	: long
-	description	: String
-	date	: Date

```

public class Projet {
    private long idProjet;
    private String description;
    private Date date;

    public long getIdProjet() {
        return idProjet;
    }

    public void setIdProjet(long id
        this.idProjet = idProjet;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String
        this.description = descriptio
    }

    public Date getDate() {
        return date;
    }

    public void setDescription(Date
        this.date = date;
    }
}

```

Si le fichier xml est correcte, alors le parseur doit
générer une classe Java

Sujet 4 : Compilateur pour un langage d'expressions mathématiques

Développez un compilateur pour un langage simplifié qui permet d'écrire et d'exécuter des expressions mathématiques. Exemple de programme :

```
a = 5;  
b = a * 2 + 10;  
afficher(b - 3);
```

Prendre en compte toutes les opérations arithmétiques ainsi que des fonctions mathématiques comme cos , sin , racine...



Sujet 5: Analyseur de fichiers de configuration

Développez un analyseur pour valider et interpréter les fichiers de configuration réseau Linux. Ces fichiers définissent les paramètres réseau pour une machine, tels que les adresses IP, les passerelles, les serveurs DNS, etc.

L'analyseur doit :

Valider la syntaxe du fichier (par exemple, vérifier que chaque section suit les règles définies).

Extraire et interpréter les données réseau (adresse IP, masque réseau, passerelle, DNS).

Générer un rapport sur la configuration validée.

Générer un fichier de configuration équivalent dans un autre format (par exemple, un fichier YAML utilisé par Netplan).

Rapport généré (fichier JSON) :

```
{
  "interfaces": [
    {
      "name": "eth0",
      "mode": "static",
      "address": "192.168.1.10",
      "netmask": "255.255.255.0",
      "gateway": "192.168.1.1",
      "dns-nameservers": [
        "8.8.8.8",
        "8.8.4.4"
      ]
    }
  ]
}
```

Sujet 6 : Générateur de scripts SQL

L'objectif est de créer un outil capable de lire une description textuelle simplifiée d'une base de données et de générer les scripts SQL nécessaires pour créer cette base. Cet outil doit prendre en charge les éléments suivants :

1. La description des tables, colonnes, et contraintes.
2. Les relations entre les tables (clés étrangères).
3. Les options avancées comme les index et les contraintes uniques.

Exemple de langage:

table utilisateurs:

- id (entier, clé primaire, auto)
- nom (texte, requis)
- email (texte, unique)
- date_creation (date, par défaut: aujourd'hui)

table articles:

- id (entier, clé primaire, auto)
- utilisateur_id (entier, requis, référence: utilisateurs.id)
- titre (texte, requis) - contenu (texte)
- date_publication (date, par défaut: maintenant)

L'objectif est de transformer ce fichier en script SQL comme suit :

```
CREATE TABLE utilisateurs (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    nom VARCHAR(255) NOT NULL,  
    email VARCHAR(255) UNIQUE,  
    date_creation DATE DEFAULT CURRENT_DATE  
);  
  
CREATE TABLE articles (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    utilisateur_id INT NOT NULL,  
    titre VARCHAR(255) NOT NULL,  
    contenu TEXT,  
    date_publication DATE DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (utilisateur_id) REFERENCES utilisateurs(id)  
);
```

Langage DSL à implémenter

Les spécifications du langage de description sont :

1. Mots-clés simples pour la déclaration des tables (table, clé primaire, référence, par défaut).

2. Types de données simplifiés : entier, texte, date, sans limites sur la longueur (le générateur choisira des valeurs par défaut).

3. Contraintes simplifiées :

- clé primaire
- auto (auto-incrémentation)
- requis (non null)
- unique
- par défaut (pour définir une valeur par défaut)
- référence (pour les clés étrangères).

Le langage doit être facile à lire et à écrire, même pour un utilisateur non expert.

Sujet 7: Compilateur pour un langage de requêtes orienté analyse de données

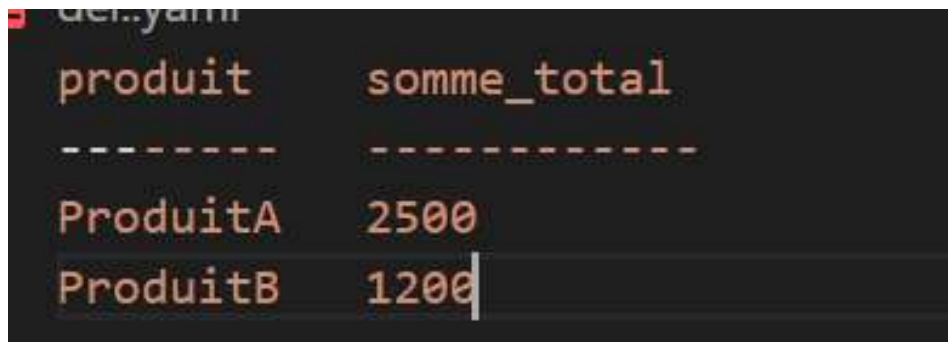
Développez un compilateur pour un langage simplifié permettant d'analyser des données dans des fichiers CSV. Le langage doit inclure des instructions pour :

- Filtrer les lignes (FILTRE),
- Sélectionner des colonnes (CHOISIR),
- Effectuer des calculs (CALCULER),
- Grouper les données (GROUPE).

Exemple de programme :

```
CHARGER "ventes.csv";  
FILTRE total > 1000;  
CHOISIR produit, total;  
GROUPE produit PAR SOMME(total);  
AFFICHER;
```

Sortie attendue (résultat d'analyse):



```
del..yann
produit      somme_total
-----
ProduitA     2500
ProduitB     1200
```

Ajouter des fonctions statistiques (MOYENNE, MAX, MIN).

Sujet 8: Analyseur d'API REST

Développez un outil qui valide la structure d'une API REST définie dans un fichier JSON ou YAML, comme une spécification OpenAPI. L'outil doit :

- Vérifier les endpoints (GET, POST, etc.),

- Valider les paramètres et les schémas de réponse,

- Générer des tests automatisés pour l'API.

Exemple d'entrée :

```
paths:
  /users:
    get:
      parameters:
        - name: id
          in: query
          required: true
          type: integer
      responses:
        200:
          description: Utilisateur trouvé
```

Sujet 9: Analyseur de logs avec génération de rapports

Créez un outil qui lit des fichiers de logs système ou applicatif, analyse leur contenu et génère des rapports.

Les logs peuvent inclure des niveaux (INFO, ERROR, DEBUG) et des timestamps.

Exemple de log :

```
2024-12-01 12:00:00 [INFO] Service démarré
2024-12-01 12:05:00 [ERROR] Connexion à la base de données échouée
2024-12-01 12:10:00 [DEBUG] Requête SQL exécutée
```

Sortie attendue :

- Nombre d'occurrences par niveau (INFO, ERROR, etc.).

- Rapport d'erreurs avec timestamps.

- Graphe de l'activité par heure.

Sujet 10: Éditeur de règles pour un système de règles métier

Développez un outil permettant de créer, valider, et exécuter des règles métier définies dans un langage spécifique (DSL). Les règles métier sont des conditions et des actions qui automatisent les processus d'entreprise, comme l'application de réductions, la validation de paiements, ou l'attribution de points de fidélité.

L'éditeur doit permettre de :

- Rédiger des règles sous forme textuelle ou via une interface utilisateur intuitive.

- Valider la syntaxe et la logique des règles définies.

- Exporter les règles au format JSON ou XML pour une intégration dans des systèmes externes.

- Simuler l'exécution des règles pour vérifier leur comportement.

Exemple de règles :

```
SI stock < 5 ET produit == "ordinateur" ALORS notifier_approvisionnement;  
SI age < 18 ALORS rejeter_transaction;
```

```
SI jour == "lundi" OU jour == "mardi" ET montant > 500  
ALORS appliquer_reduction 5%;
```

```
SI montant > 1000 ALORS appliquer_reduction 10%;  
SI type_client == "VIP" ALORS ajouter_points 50;
```

Sujet 11: Générateur de documentation à partir de commentaires



Développez un outil qui extrait les commentaires Javadoc des fichiers source Java, valide leur syntaxe, et génère automatiquement une documentation structurée en différents formats comme HTML, Markdown, ou JSON. L'objectif est de fournir une solution simple pour transformer le contenu des commentaires Javadoc en une documentation lisible et bien structurée.

Les commentaires Javadoc suivent une structure standard et contiennent des annotations spécifiques comme `@param`, `@return`, et `@throws`, qui doivent être traitées correctement.

Sortie Html

```
<h1>Classe : Client</h1>
<p><strong>Description :</strong> Classe représentant un client dans le système.</p>
<p><strong>Auteur :</strong> John Doe</p>
<p><strong>Version :</strong> 1.0</p>

<h2>Méthodes</h2>
<ul>
  <li><strong>Constructeur :</strong> <code>Client(String name, String email)</code>
    <ul>
      <li><strong>@param name :</strong> Le nom du client.</li>
      <li><strong>@param email :</strong> L'adresse email du client.</li>
    </ul>
  </li>
  <li><strong>getName</strong> : Retourne le nom du client.
    <ul>
      <li><strong>@return :</strong> Le nom du client.</li>
    </ul>
  </li>
  <li><strong>setName</strong> : Modifie le nom du client.
    <ul>
      <li><strong>@param name :</strong> Le nouveau nom du client.</li>
    </ul>
  </li>
</ul>
```

Sortie Markdown

```
# Classe : Client
**Description :** Classe représentant un client dans le système.
**Auteur :** John Doe
**Version :** 1.0

## Méthodes
- **Constructeur :** `Client(String name, String email)`
  - **@param name :** Le nom du client.
  - **@param email :** L'adresse email du client.

- **getName** : Retourne le nom du client.
  - **@return :** Le nom du client.

- **setName** : Modifie le nom du client.
  - **@param name :** Le nouveau nom du client.
```

Entrée Client.java

```
/**
 * Classe représentant un client dans le système.
 * @author John Doe
 * @version 1.0
 * @since 1.0
 */
public class Client {
    private String name;
    private String email;

    /**
     * Constructeur de la classe Client.
     *
     * @param name Le nom du client.
     * @param email L'adresse email du client.
     */
    public Client(String name, String email) {
        this.name = name;
        this.email = email;
    }

    /**
     * Retourne le nom du client.
     *
     * @return Le nom du client.
     */
    public String getName() {
        return name;
    }

    /**
     * Modifie le nom du client.
     *
     * @param name Le nouveau nom du client.
     */
    public void setName(String name) {
        this.name = name;
    }
}
```

Sujet 12: DSL pour des scénarios de jeu vidéo

Concevez un DSL pour définir des scénarios interactifs dans un jeu vidéo, comme des dialogues et des quêtes.

```
scène 1:
  dialogue:
    personnage1: "Bonjour, aventurier !"
    personnage2: "Je cherche une quête."
  quête:
    objectif: "Trouver 10 herbes médicinales."
```

Sortie attendue :

Génération d'un fichier JSON ou XML interprété par le moteur de jeu.

Sujet 13: DSL pour un langage de commande domotique

Développez un DSL pour configurer des scénarios de commande domotique.

```
allumer lumière salon à 18:00;  
éteindre chauffage à 23:00;
```

Sortie attendue :

Génération de commandes interprétées par un système domotique.