L'objectif de ce projet est de créer un interpréteur pour un langage permettant d'automatiser l'écriture de pages web en HTML/CSS. Les spécificités du langage sont détaillées dans la suite du sujet.

Le projet peut se faire seul ou en binôme et est à rendre au plus tard le dimanche 23 février sur le dépôt "Projet 2024-2025" sur moodle.

Généralités

Le projet est divisé en deux parties : la reconnaissance du langage et l'interprétation du langage. La reconnaissance du langage sera effectuée dans un premier temps. Les explications nécessaires à l'interprétation du langage seront fournies le lundi 6 janvier lors d'une mise à jour du sujet.

Pour réaliser votre projet vous pouvez récupérer l'archive `projet.zip` sur moodle contenant la base du projet. Une archive `exemple.zip` vous sera proposé le 6 janvier contenant des fichiers devant être interprétés et le résultat associé.

Prérequis

Afin de pouvoir compiler le projet il est nécessaire d'avoir installé les paquets suivant (sur Linux) :

- flex
- libfl-dev
- bison

Pour Mac OS X les paquets doivent être similaires.

Pour compiler nous utilisons cmake, il est conseillé de le faire à partir du terminal en créant un répertoire build à l'intérieur du projet puis de faire :

```
cmake ..
  make
./projet
```

Attention donc si vous modifiez le répertoire `expressions` à bien recompiler entièrement votre projet dans un terminal.

Au cours du projet vous devez stocker les différentes instructions dans une structure de données adaptée avant de produire un fichier de sortie.

Langage

Instructions simples

Les instructions de notre langage se terminent par un retour à la ligne (pas de symbole de fin d'instruction).

Il est possible de créer des blocs pour décrire le contenu de la page :

- Des titres : `!T texte`
- Des sous-titres : `!TT texte` (Le nombre de T indique le niveau du sous-titre)
- Des paragraphes : `!P texte`
- Des images : `!I texte`

Les textes peuvent faire plusieurs lignes et sont encadrés par des simples quotes `'` on pourra utiliser le caractère d'échappement pour intégrer des simples quotes au texte. Attention il faudra penser à ne pas afficher le caractère d'échappement dans la sortie. Un texte peut inclure des retours à la ligne sans que l'instruction se termine.

Pour les images le texte correspond à la source de l'image, on utilisera les chemins sous le format UNIX.

Paramètres de la page

Il est possible de paramétrer la page web afin d'intégrer les métadonnées classiques (titre, encodage des caractères, etc…) pour cela on utilise les commandes suivates :

- Définition d'une propriété : `@DEFINE (propriété) {valeur}`
- Définition d'un titre pour la page : `@TITREPAGE texte`

Les propriétés possibles sont les suivantes :

- `encodage` avec pour valeur une chaîne de caractères représentant l'encodage (utf-8, ISO-8859-1, …)
- `icone` avec pour valeur le chemin vers l'image à utiliser
- `css` avec pour valeur le chemin vers la feuille de style
- `langue` avec pour valeur la langue utilisée (fr, en, …)

Les commentaires

Notre langage permet d'écrire des commentaires qui seront visibles dans la sortie produite

- `%% texte` permet d'écrire le commentaire texte sur une seule ligne
- `%%% texte %%%` permet d'écrire le commentaire texte sur plusieurs lignes

Les attributs de blocs

Il est possible de paramétrer les attributs des différents blocs de la façon suivante :

```
!T [attribut1 : valeur, attribut2 : valeur, ... attribut n : valeur] texte
```

Les attributs courants sont :

- `largeur` : valeur entière (suivi ou non de `px`)
- `hauteur` : valeur entière (suivi ou non de `px`)
- `couleurTexte` : valeur couleur (défini juste en dessous)
- `couleurFond` : valeur couleur (défini juste en dessous)
- `opacité` : valeur entière (suivi ou non de `%`)

Les couleurs possèdent deux formats :

- `rgb(R,G,B)` (avec R, G et B des valeurs entières entre 0 et 255)
- `#RRGGBB` (avec RR, GG et BB des valeurs hexadécimales)

Propriétés de style

Nous donnons aussi la possibilité de définir le style pour un même type de bloc en précisant le nom du bloc et les propriétés à appliquer :

- Définition d'un style :

```
@STYLE ( bloc ) [
  attribut1 : valeur
  attribut2 : valeur
  ...
  attribut n : valeur
]
```

Les blocs pouvant être affectés par un style sont :

- `page` (affecte un style à tout le document)
- `paragraphe` (affecte un style aux paragraphes)
- `titre1` (affecte un style aux titres de niveau 1 à 9)

Utilisation des variables

Vous pouvez utiliser trois types de variables :

- Des entiers
- Des couleurs
- Des éléments de contenu
- Un style

La déclaration des variables se fait à l'aide du nom de la variables. Le nom d'une variable ne peut être associé à différents types.

```
maVariableEntiere = 5
maVariableCouleur = #FF0000
maVariableBloc = !P "Mon paragraphe"
monStyle = [attribut1 : valeur, attribut2 : valeur, ... attribut n : valeur ]
```

Les sélecteurs

Tous les éléments de la page sont accessibles à l'aide d'un sélecteur (cela signifie que vos éléments doivent être stockés dans une structure de données adaptée). On peut donc accéder à un bloc et le stocker dans une variable de cette façon :

```
maVariableBloc = !P[0]
```

Ce qui signifie que maVariableBloc sélectionne le premier paragraphe de la page. On peut ensuite modifier les paramètres d'un bloc à l'aide de ses membres :

```
maVariableBloc.couleurFond = #0000ff
```

On pourra modifier l'ensemble des attributs : `largeur`, `hauteur`, `couleurTexte`, `couleurFond`, `opacité` mais aussi appliquer un style :

```
maVariableBloc.style = monStyle
```

Les conditionnelles

Note langage propose de gérer les instructions conditionnelles sous la forme :

```
SI <condition> :
instruction1
instruction2
instruction3
FINSI
```

L'indentation n'a pas d'importance. On peut aussi utiliser le mot-clé SINON :

```
SI <condition> :
instruction 1
instruction 2
SINON :
instruction 3
instruction 4
FINSI
```

Il est aussi possible d'imbriquer les conditions en pensant à fermer chaque `SI` par un `FINSI`

```
SI <condition> :
instruction 1
instruction 2
SINON :
instruction 3
SI <condition> :
instruction 4
FINSI
FINSI
```

Les conditions possibles seront détaillées dans la deuxième partie du projet début janvier.

Les boucles

On utilisera un seul type de boucles qui sera une boucle `POUR`.

```
POUR i [0,10] +1 :
instruction 1
instruction 2
...
instruction n
FINI
```

Interprétation du langage

Afin que vous vous concentriez sur la reconnaissance du langage nous omettons cette section pour l'instant. Une seconde version du sujet avec cette partie vous sera communiquée. Cette nouvelle version vous présentera la structure du langage de sortie (qui sera le format HTML/CSS). Pour cette première partie vous pouvez construire entièrement votre grammaire et la structure de données associée. Il ne restera ensuite plus qu'à écrire les instructions dans la sortie standard ou dans un fichier HTML pour la sortie.

Notation

Votre projet doit se constituer ainsi :

- Analyseur lexical (scanner.ll)
- Analyseur syntaxique (parser.yy)
- Structure de données (fichiers à créer et utilisation des expressions)
- Écriture d'un fichier HTML en sortie

Il est important d'avoir une structure de données intermédiaire afin de pouvoir avoir une entrée et une sortie différente. Vous devez donc modéliser votre page dans cette structure intermédiaire. Le projet sera noté principalement sur votre parser et votre structure de données. La partie analyse lexicale (élaboration des expressions régulières dans le fichier scanner.ll) ne compte que pour une petite partie de la note finale. La sortie de votre parser (écriture du fichier HTML) sera comptée uniquement en points bonus mais vous servira à vérifier que votre programme fonctionne correctement. La majorité des points est basée sur votre analyseur syntaxique (fichier parser.yy) et votre structure de données intermédiaire.

Un projet ne contenant que l'analyseur syntaxique et l'analyseur lexical ne sera pas bien noté il faut ABSOLUMENT avoir une structure de données intermédiaire. Un bon parser nécessite un arbre de syntaxe abstraite (AST) et un algorithme permettant de dérouler cet arbre afin d'exécuter le programme dans le bon ordre.