Projet : Création et rendu d'images en SVG

Consignes :

- Lisez bien tout le sujet avant de commencer à coder.
- Déposez avant le **vendredi 28 avril à 23h59** une archive au format `.tar.gz`, contenant votre code pour les trois premières parties, vos fichiers de test éventuels et un rapport au **format PDF**, dans le dépôt `proc_fisa_2223` sur `http://exam.ensiie.fr`.

L'objectif de ce projet est de développer un outil permettant de définir et de transformer des images composées de diverses formes géométriques. Cet outil prendra en entrée une suite de commandes, et produira sur demande un export de l'image courante au format SVG.

Le sujet est composé de quatre parties :

1. la mise en place des bases du projet (syntaxe et gestion des premières commandes),
2. le rendu en SVG,
3. la gestion du reste des commandes,
4. une liste de plusieurs améliorations possibles.

Vous devez faire au minimum les trois premières parties, et il faudra aborder certaines questions de la quatrième partie pour avoir une (très) bonne note.

1 Échauffement avec les premières commandes

L'objectif de cette partie est de poser les bases du projet, c'est-à-dire de produire :
- un analyseur lexical (fichier `.l`) simple,
- un analyseur syntaxique (fichier `.y`) minimaliste,
- une fonction `main` pour lier le tout.

1.1 Définitions préliminaires

Le langage utilisé pour définir et manipuler des images comporte plusieurs mots-clés. Ces mots-clés, écrits tout en majuscules dans ce document et dont l'usage est spécifié par la suite, sont les suivants :

ALL	DONE	MOVE	SET
AT	DUMP	NOFILL	TEXT
CIRCLE	ELLIPSE	POLYGON	THICKNESS
COPY	FILL	RADIUS	VISIBLE
CREATE	FONTSIZE	RENAME	WITH
DELETE	FOREACH	RECTANGLE	ZOOM
DESELECT	INVISIBLE	ROTATE	
DO	LINE	SELECT	

Votre programme devra reconnaître ces mots-clés, qu'ils soient écrits en majuscules, en minuscules, ou avec un mélange de majuscules et minuscules.

Dans ce projet, un *nom* est une suite de caractères[1] commençant par une lettre (minuscule ou majuscule) et composé uniquement de lettres, chiffres, `-` et `_`. Une *chaîne de caractères* est une suite de caractères entourés par des `"` et ne contenant pas le symbole `\`.

Une *couleur* est une suite de caractères, constituée du symbole `#` suivi de 3 ou 6 chiffres hexadécimaux (permettant de choisir la quantité de rouge, de vert et de bleu, sur 1 ou 2 chiffres hexadécimaux respectivement).

1. Si cela vous simplifie la vie, vous pouvez supposer qu'un nom a au plus 20 caractères, ou encore tronquer les noms pour ne conserver que les 20 premiers caractères.

Les *coordonnées* sont des couples d'entiers, entre parenthèses et séparés par une virgule. Les *dimensions* sont un couple d'entiers séparés par la lettre `x`. On utilisera `<num>` pour désigner un nombre entier, et `<real>` pour désigner un nombre à virgule flottante. Avec ces notations, une dimension peut donc être notée `<num>x<num>`.

Une *figure* est un élément à afficher. Chaque figure possédera au moins un *nom* unique et les *coordonnées* de son centre (couple d'entiers entre parenthèses et séparés par une virgule). D'autres valeurs viendront s'ajouter selon le type de figure. Enfin, une *image* est un ensemble de figures.

1.2 La commande CREATE

Afin de définir une figure, nous allons utiliser la commande CREATE, qui possède pour le moment les syntaxes suivantes :

- `CREATE CIRCLE <name> AT <coord> RADIUS <num>`
- `CREATE CIRCLE <name> RADIUS <num> AT <coord>`

Nous nous contentons ici de créer des cercles. Les autres types de figures seront traités plus loin dans le sujet, et viendront étoffer la syntaxe de CREATE.

Question 1. Écrivez un analyseur lexical et un analyseur syntaxique afin de reconnaître une suite de commandes CREATE valides saisies sur l'entrée standard.

note : Il s'agit juste, dans un premier temps, d'afficher un message d'erreur en cas de syntaxe invalide, et de ne rien faire si la commande saisie est valide.

1.3 La commande DUMP

La commande DUMP permet d'exporter l'image courante. Elle possède les syntaxes suivantes :

- `DUMP`
- `DUMP <string>`

L'appel à la commande DUMP sans argument provoquera l'affichage sur la sortie standard de toutes les figures constituant l'image courante. Chaque figure donnera lieu a une ligne de texte, donnant le plus d'informations possibles sur la figure (tout en restant lisible). Cette syntaxe servira principalement à récupérer les noms exacts des figures, et éventuellement à déboguer votre projet.

Si la commande DUMP est suivie d'une chaîne de caractère, on considérera que cette chaîne contient le chemin vers un fichier accessible en écriture. On remplira ce fichier avec du code au format SVG correspondant à l'image courante. L'image pourra alors être affichée en ouvrant le fichier produit dans votre navigateur web favori.

Question 2. Modifiez vos analyseurs lexical et syntaxique afin de reconnaître les commandes DUMP.

Question 3. Dans votre fichier `.y`, implantez les actions nécessaires pour traiter correctement la commande CREATE, ainsi que la commande DUMP sans arguments.

notes :

- L'export au format SVG (seconde syntaxe de DUMP) fera l'objet de la partie 2 ;
- L'ordre d'affichage des figures n'a pas d'importance lors d'un appel à DUMP sans argument ;
- On rappelle qu'il ne peut pas y avoir deux figures avec le même nom. Créer une figure dont le nom est déjà utilisé devra provoquer une erreur ;
- Pour afficher un message d'erreur, on pourra faire appel à la fonction `yyerror` ou directement à `fprintf`. Par ailleurs, le code `YYERROR;` provoque une erreur au sein de `bison`, mais sans afficher de message d'erreur. Idéalement, il faudra donc utiliser conjointement `yyerror/fprintf`, `YYERROR` et le symbole non-terminal `error` vu en TP ;
- Il est vivement recommandé d'écrire le code relatif à la création et à la manipulation de figures dans un fichier `.c` à part.

1.4 La commande DELETE

La commande DELETE permet, étant donné un nom, de supprimer la figure correspondante (si une telle figure existe). Sa syntaxe est la suivante :

— DELETE <name>

Question 4. Modifiez votre code afin de gérer correctement la commande DELETE.

1.5 La commande RENAME

La commande RENAME permet de changer le nom d'une figure. Sa syntaxe est la suivante :

— RENAME <name> WITH <name>

Le premier nom doit correspondre à une figure existante. Le second champ <name> fournit le nouveau nom de la figure, à condition que ce nom ne soit pas déjà utilisé par ailleurs. En cas de problème, il convient d'afficher un message adéquat et de provoquer une erreur.

Question 5. Modifiez votre code afin de gérer correctement la commande RENAME.

2 Export au format SVG

L'objectif de cette partie est d'écrire le code permettant de convertir l'image courante en fichier SVG. En termes de commandes, il s'agit donc en fait de gérer DUMP appelé avec un argument.

2.1 Premières images en SVG

Pour une petite introduction au format SVG, vous pouvez consulter les sites suivants :

— https://www.w3schools.com/graphics/svg_intro.asp

— https://developer.mozilla.org/fr/docs/Web/SVG

ainsi que l'exemple fourni sur la page web du cours (http://web4.ensiie.fr/~christophe.mouilleron/teaching/PROC/example.svg). Je vous invite d'ailleurs à consulter le contenu de ce fichier exemple.

Les images que nous allons produire auront pour taille[2] 800x600. Pour des raisons de lisibilité, on impose que toute image dispose (au minimum) d'un contour noir. Ainsi, le code SVG correspondant à une image sans figure est le suivant :

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
     width="800" height="600" viewBox="0 0 800 600">
    <rect x="0" y="0" width="800" height="600" fill="none" stroke="black" />
</svg>
```

Les codes SVG des différentes figures devront alors être placés juste après la balise <rect>. On précise enfin que les coordonnées sont quelconques. Il est donc tout à fait possible qu'une figure n'apparaisse que partiellement, voire pas du tout, lorsqu'on ouvre le fichier .svg produit dans un navigateur web.

Question 6. Écrivez le code permettant de générer du SVG pour les cercles, et complétez le support de la commande DUMP.

2.2 Un peu de style

En testant votre code, vous remarquerez que les cercles produits jusqu'à présent sont en fait des disques (pleins) noirs. Pour modifier l'apparence de ces cercles, nous allons ajouter la commande SET, dont la syntaxe est la suivante :

— SET <name> <option>

— SET <name> <option_1> <option_2> ... <option_k>

avec la contrainte $k \geq 2$ pour la deuxième syntaxe.

2. Cette taille est suffisamment petite pour être affichée sur n'importe quel écran d'ordinateur, et suffisamment grande pour permettre des dessins précis malgré le choix de se restreindre à des coordonnées entières.

Le nom permet de désigner la figure précise dont on souhaite modifier l'apparence. Et les options à gérer dans un premier temps sont résumées dans le tableau suivant.

syntaxe	effet
`<color>`	définit la couleur du contour
`THICKNESS <num>`	définit l'épaisseur du contour
`FILL WITH <color>`	définit le couleur de remplissage
`NOFILL`	demande de ne pas remplir la figure
`INVISIBLE`	demande de ne pas afficher la figure
`VISIBLE`	demande d'afficher la figure (annule `INVISIBLE`)

Par défaut, les figures sont supposées visibles, avec un contour et un remplissage en noir.

Question 7. Modifiez la grammaire de votre fichier `.y` pour reconnaître les commandes `SET` valides.

Question 8. Faites en sorte que le génération du code SVG prenne bien en compte les éléments de style définis grâce à la commande `SET`.

3 Reste des commandes à gérer

Cette section présente la liste des commandes qui n'ont pas encore abordées et que votre programme devra impérativement gérer.

3.1 Autres types de figures

En plus des cercles, nous allons gérer trois autres types de figures, que nous pourrons créer en utilisant la syntaxe suivante :

- `CREATE LINE <name> <coord_1> <coord_2>`
- `CREATE RECTANGLE <name> AT <coord> <dimension>`
- `CREATE RECTANGLE <name> <dimension> AT <coord>`
- `CREATE TEXT <name> AT <coord> <string>`
- `CREATE TEXT <name> <string> AT <coord>`

Le type `LINE` correspond à une ligne droite entre les deux coordonnées fournies. Le type `RECTANGLE` correspond à un rectangle donné par son centre et ses dimensions (largeur et hauteur, dans cet ordre). Enfin, le type `TEXT` permettra d'afficher le texte donné dans le champ `<string>`.

Question 9. Ajoutez le support de ces trois types de figures.

Question 10. Pour le champ texte, il est utile de pouvoir préciser la taille à utiliser. Pour cela, ajoutez le support de l'option `FONTSIZE <num>`, valide uniquement dans le cas d'une figure de type texte.

3.2 Transformation sur les figures

À l'issue d'un rendu en SVG, on peut être amené à vouloir changer une partie de l'image. Pour ce faire, on va proposer un nouvel ensemble de commandes. D'abord, nous avons besoin de choisir les figures sur lesquelles on va opérer une transformation. C'est le rôle des commandes `SELECT` et `DESELECT`, dont la syntaxe est la suivante :

- `SELECT <name>`
- `SELECT <name_1> , <name_2> , ... , <name_k>`
- `SELECT ALL`
- `DESELECT <name>`
- `DESELECT <name_1> , <name_2> , ... , <name_k>`
- `DESELECT ALL`

On peut ainsi sélectionner (ou désélectionner) une ou plusieurs figures, en donnant leurs noms séparés par une virgule. On peut aussi utiliser le mot clé `ALL` à la place des noms, pour désigner directement l'intégralité des figures.

Après sélection des certaines figures, nous pourrons utiliser les commandes suivantes pour appliquer des transformations :

— `MOVE <coord>`
— `ZOOM <real>`
— `ROTATE <num>`

À chaque fois, la transformation ne s'appliquera que sur les figures sélectionnées. La transformation `MOVE` consiste à appliquer une translation de vecteur indiqué par le champ `<coord>`, qui peut être vu comme le vecteur (d_x, d_y). La transformation `ZOOM` permet d'agrandir ou de rétrécir une figure. Mathématiquement, il s'agit d'une homothétie par rapport au centre de la figure et de facteur indiqué par le champ `<real>`. Enfin, la transformation `ROTATE` permet de faire pivoter la figure autour de son centre d'un certain angle exprimé en degrés via le champ `<num>`.

Question 11. Ajoutez le support des commandes `SELECT` et `DESELECT`.

Question 12. Ajoutez le support des trois transformations `MOVE`, `ZOOM` et `ROTATE`.

note : Comme on travaille en coordonnées entières, on pourra au besoin arrondir les valeurs calculées (typiquement lors d'un zoom).

4 Quelques améliorations possibles

4.1 Amélioration de la syntaxe existante

La syntaxe proposée dans ce sujet peut être grandement améliorée. En particulier, on pourrait mettre en place :

— la définition des éléments de style directement dans la commande `CREATE`
— la possibilité de mélanger les éléments de définition et les éléments de style au sein d'une commande `CREATE`, rendant par exemple la commande suivante valide

```
CREATE CIRCLE AT (100,100) mycicrle #f00 FILL WITH #ff0 RADIUS 50
```

— la possibilité lors d'une transformation de donner directement les noms des figures concernées, rendant par exemple la commande suivante valide

```
MOVE cercle1,rectangle4 (10,20)
```

— la possibilité d'appliquer la commande `SET` à toutes les figures (avec `ALL` à la place du nom de figure), à toutes les figures sélectionnées (en omettant le nom de figure), ou à plusieurs figures en donnant leurs noms séparés par des virgules
— la copie d'une figure via la commande `COPY <name_1> <name_2>`, créant une figure de nom `<name_2>` identique à la figure nommée `<name_1>`.

Question 13. Implantez tout ou partie de ces améliorations de syntaxe.

4.2 Mode non-interactif

Jusqu'à présent, les commandes sont saisies sur l'entrée standard, en mode interactif. Si le programme est lancé avec un nom de fichier en argument, on traitera plutôt ce dernier en mode non-interactif. Ce mode possède les particularités suivantes :

— Les commandes sont lues depuis un fichier, et plus depuis l'entrée standard.
— On peut mettre plusieurs commandes sur une seule ligne, à condition de les séparer par un `;`.
— Désormais, toute erreur (comme par exemple la suppression d'une figure qui n'existe pas) met immédiatement fin au programme.
— En cas d'erreur, il faudra afficher sur la sortie d'erreur un message contenant le numéro de ligne à laquelle cela s'est produit. Autant que possible, on essaiera aussi de donner des détails sur la nature de l'erreur.

Question 14. Implantez le mode non-interactif.

4.3 Support pour les ellipses et les polygones

Le format SVG propose deux autres formes géométriques que nous ne gérons pas encore. Pour remédier à ce problème, nous introduisons la syntaxe suivante :

- `CREATE ELLIPSE <name> AT <coord> RADIUS <dimension>`
- `CREATE ELLIPSE <name> AT RADIUS <dimension> AT <coord>`
- `CREATE POLYGON <name> <coord_1> <coord_2> ... <coord_k>`

Pour le type `ELLIPSE`, le champ `<dimension>` fournit le rayon r_x selon l'axe des x, puis le rayon[3] r_y selon l'axe des y. Pour le type `POLYGON`, il faudra avoir $k \geq 3$ points (donc au moins un triangle).

Question 15. Ajoutez à votre programme le support des figures de type `ELLIPSE`.

Question 16. Ajoutez à votre programme le support des figures de type `POLYGON`.

4.4 Génération par lots [⋆]

Afin de créer plusieurs figures similaires en une fois, on se propose de mettre en place la syntaxe `FOREACH <indices> DO <commande> ; ... ; <commande> DONE`.

Ici, `<indices>` sera une suite d'entiers séparés par des virgules, le tout entre les symboles `{` et `}`. On exécutera alors la liste des commandes pour chacun des ces entiers successivement.

Afin d'utiliser l'entier courant au sein des commandes, on va s'autoriser à écrire des calculs. Ces calculs seront entourés des symboles `{` et `}`, et pourront contenir la lettre `x` désignant l'entier courant, des constantes entières, les opérations arithmétiques classiques[4] (`+`, `-`, `*` et `%` pour le modulo), et des parenthèses pour imposer des priorités. Ces calculs pourront être utilisés à l'intérieur d'un nom, à l'intérieur d'un chaîne de caractères, ou en remplacement d'une valeur entière.

Question 17. Implantez et testez cette nouvelle syntaxe.

4.5 Autres améliorations

En plus de ce qui a été mentionné précédemment, voici une liste (non exhaustive) de points qui méritent votre attention :

- la qualité des messages d'erreur,
- la simplicité et la lisibilité de la grammaire définie dans le fichier `.y`,
- la qualité générale de votre code (structures de données efficaces, modularité, factorisation).

3. Bien sûr, si on choisit $r_x = r_y$, la figure obtenue est visuellement un cercle de rayon r_x.
4. La division a été volontairement omise ici.