Théorie des Langages et Compilation
TP Projet – *Langage de commande de dessin.*

Vous remettrez un seul projet, sans duplication ou autre indication de type exo 1, .... Vous ajouterez un fichier `ReadMe.txt` qui contiendra vos commentaires sur le projet et en particulier jusqu'à quel exercice vous êtes allés.

L'objectif de ce projet est de créer un langage capable de donner des ordres de dessins au travers d'un langage dédié.

# Présentation

Nous voulons modéliser un langage pour contrôler un robot capable de dessiner dans une zone à dessin. Voici les actions réalisables de notre robot :

- Nous pouvons baisser le crayon (pour écrire)
- Nous pouvons lever le crayon (pour ne plus écrire)
- Nous pouvons nous déplacer dans la zone de dessin. Ainsi, si le crayon est baissé nous dessinons et si le crayon est levé alors le bras se déplace et rien n'est dessiné.
- Nous pouvons changer la couleur du crayon. Une fois choisie, la couleur reste jusqu'au prochain changement.
- La couleur par défaut est le noir. Et la zone de dessin est une autre couleur (par exemple le blanc).
- Les coordonnées de la zone de dessin est uniquement sur les entiers.
- Nous avons aussi la possibilité de dessiner des formes basiques : les lignes et les rectangles.

La spécificité est que nous n'allons pas sauvegarder l'état des pixels mais bien la suite des commandes réalisées, et nous allons ajouter en plus les éléments suivants :

- La possibilité de déclarer des variables entières.
- La possibilité de faire des calculs sur ces entiers mélangeant entiers et littéraux entiers.
- La possibilité de faire des boucles, où nous donnerons la variable d'itération, la valeur initiale (possiblement sous forme de calcul) et la valeur finale incluse( possiblement sous forme de calcul).
- Nous utiliserons l'opérateur $<-$ pour l'affectation. Exemple : x $<-$ 3 + 2;
- Toutes les lignes de notre commande se termine par un point virgule (cette contrainte est faite pour vous simplifier la vie si vous n'en voyez pas l'utilité, vous pouvez la laisser de coté en croisant les doigts que je suive votre pensée).

# Exercices

### Exercice 1: Analyse lexical & Conception

Proposez un lexeur qui regroupe l'ensemble des lexèmes que vous aurez choisi. Nous vous demandons d'éviter d'exploiter des opérateurs/symboles pour les structures de flot de contrôle usuel pour avoir une syntaxe facilement compréhensible pour un utilisateur extérieur français (viser enfant au collège).

Vous proposerez dans votre intuition de syntaxe deux fichiers d'exemple :

- Un fichier réalisant un quadrilatère d'une couleur et 2 lignes de couleur différentes pour former une croix au centre du quadrilatère.
- Un fichier qui réalise un quadrillage en utilisant une boucle et exploitant des variables bien senties.

### Exercice 2: Reconnaissance du langage

Réalisez un programme qui reconnaît la syntaxe de votre programme. En particulier votre programme devra répondre si oui ou non le programme fourni respecte ou non votre syntaxe.

### Exercice 3: Construction de l'arbre sémantique

En vous inspirant de ce qui a été fait en TP, réalisez un arbre de terme qui reprend les possibilités de votre langage et modifiez votre programme précédent pour construire l'arbre syntaxique de façon cohérente.

Au sein de l'arbre, vous vérifierez qu'il n'y a pas d'affectation sur les variables d'itérations. Et vous indiquerez ça sur la sortie de votre programme.

### Exercice 4: Amélioration

Proposez 1 ou 2 amélioration de la syntaxe (selon la difficulté apparente de vos modifications) et adaptez votre solution en conséquence.

### Exercice 5: Bonus : Exploitation de l'arbre

En vous inspirant de la mécanique du visiteur et d'une bibliothèque d'image 2D (je vous propose : `http://cimg.eu/`, mais vous pouvez prendre ce que vous voulez de simple), Vous appliquerez les commandes de votre fichier pour obtenir une image (dans un format standard, bmp, png, jpg selon les possibilités).

Voici un extrait de code d'utilisation de CImg qui réalise quelques traitements divers :

```
#include "CImg.h"
using namespace cimg_library;
int main()
{
	int sizeX = 320, sizeY = 240;

	CImg<unsigned char> img(sizeX, sizeY, 1, 3);
	img.fill(255); // on remplit en blanc
	// ci dessous exemple de couleur RGB
	const unsigned char red[] = { 255, 0 , 0 };
	const unsigned char green[] = { 0, 255 , 0 };
	const unsigned char blue[] = { 0, 0 , 255 };
	// action
	img.draw_line(0, 0, 10, 10, green);
	img.draw_rectangle(10, 10, 100, 100, blue);
	img.save_bmp("toto.bmp");
	// j'ai eu des soucis avec d'autre formatle bmp est gros mais
	// ca compile, ca marche

	// optionnel mecanique d'affichage
	// pendant votre programme.
	CImgDisplay disp;
	disp.display(img);
	while (!disp.is_closed()) {
		// boucle des evenements
		if (disp.key() == cimg::keyQ)
		disp.close();
		disp.wait();
	}

	return 0;
}
```