Le langage $draw^{++}$
ING1 apprentis GI
Projet C

2024

1 Description du projet

Le but de ce projet est de concevoir un langage, intitulé $draw^{++}$, qui permet de dessiner des formes quelconques sur un écran à partir d′instructions à définir. Un répertoire d'instructions doit être construit en premier lieu. Ce répertoire doit être compact, complet et non redondant. Vous aurez deux types d'instructions : les élémentaires et les évoluées.

1.1 Instructions élémentaires

Parmi les instructions élémentaires à proposer, nous avons :

— Créer un curseur. Vous pouvez en avoir plusieurs. Un curseur a une position actuelle sur l'écran, souvent représentée par des coordonnées (x, y). Il peut être visible ou non sur l'écran.

— Affecter une couleur ou une épaisseur à un curseur.

— Faire avancer (un saut) un curseur de manière relative exprimée en pixels.

— Faire pivoter un curseur d'une quantité relative exprimée en degrés.

— Dessiner, pour un curseur donné, une forme (segment, carrée, cercle, point, arc, ...). Les caractéristiques des formes doivent être spécifiées comme opérandes des instructions.

— Animer un dessin.

— Et d'autres instructions à proposer.

1.2 Instructions évoluées

Parmi les instructions évoluées à proposer, nous avons :

— Instruction d'assignation : Affectation de valeurs à des variables .

— Instructions de Bloc : Regroupement d'instructions en une seule unité.

— Instructions Conditionnelles : if, else pour exécuter du code en fonction de conditions.

— Instructions Répétitives : Boucles for, while, do-while pour répéter des instructions plusieurs fois.

2 Grammaire de *draw*$^{++}$

La conception de votre langage nécessite la définition de sa grammaire. Cela vous permettra de spécifier clairement les caractéristiques de votre langage. La grammaire d'un langage de programmation est un ensemble de règles définissant la syntaxe et la sémantique des programmes écrits dans ce langage. Elle permet au compilateur ou l'interpréteur à "comprendre" et exécuter les instructions. Grâce à la grammaire d'un langage, les compilateurs ou interpréteurs peuvent signaler les erreurs, aidant ainsi le développeur à corriger ses codes rapidement. La Figure 1 donne un aperçu de la grammaire du langage Java.

```
Blocks and Commands
    <block> ::=  <block statements>?
    <block statements> ::= <block statement> | <block statements> <block statement>
    <block statement> ::= <local variable declaration statement> | <statement>
    <local variable declaration statement> ::= <local variable declaration> ;
    <local variable declaration> ::= <type> <variable declarators>
    <statement> ::= <statement without trailing substatement> | <labeled statement> |
        <if then statement> | <if then else statement> | <while statement> |
        <for statement>
    <statement no short if> ::= <statement without trailing substatement> |
        <labeled statement no short if> | <if then else statement no short if> |
        <while statement no short if> | <for statement no short if>
    <statement without trailing substatement> ::= <block> | <empty statement> |
        <expression statement> | <switch statement> | <do statement> | <break statement> |
        <continue statement> | <return statement> | <synchronized statement> |
        <throws statements> | <try statement>
    <empty statement> ::= ;
    <labeled statement> ::= <identifier> : <statement>
    <labeled statement no short if> ::= <identifier> : <statement no short if>
    <expression statement> ::= <statement expression> ;
    <statement expression> ::= <assignment> | <preincrement expression> |
        <postincrement expression> | <predecrement expression> |
        <postdecrement expression> | <method invocation> |
        <class instance creation expression>
    <if then statement>::= if ( <expression> ) <statement>
    <if then else statement>::= if ( <expression> ) <statement no short if> else <statement>
```

```
    <if then else statement no short if> ::= if ( <expression> ) <statement no short if>
        else <statement no short if>
    <switch statement> ::= switch ( <expression> ) <switch block>
    <switch block> ::=  <switch block statement groups>? <switch labels>?
    <switch block statement groups> ::= <switch block statement group> |
        <switch block statement groups> <switch block statement group>
    <switch block statement group> ::= <switch labels> <block statements>
    <switch labels> ::= <switch label> | <switch labels> <switch label>
    <switch label> ::= case <constant expression> : | default :
    <while statement> ::= while ( <expression> ) <statement>
    <while statement no short if> ::= while ( <expression> ) <statement no short if>
    <do statement> ::= do <statement> while ( <expression> ) ;
    <for statement> ::= for ( <for init>? ; <expression>? ; <for update>? ) <statement>
    <for statement no short if> ::= for ( <for init>? ; <expression>? ; <for update>? )
        <statement no short if>
    <for init> ::= <statement expression list> | <local variable declaration>
    <for update> ::= <statement expression list>
    <statement expression list> ::= <statement expression> |
        <statement expression list> , <statement expression>
    <break statement> ::= break <identifier>? ;
    <continue statement> ::= continue <identifier>? ;
    <return statement> ::= return <expression>? ;
    <throws statement> ::= throw <expression> ;
    <synchronized statement> ::= synchronized ( <expression> ) <block>
    <try statement> ::= try <block> <catches> | try <block> <catches>? <finally>
    <catches> ::= <catch clause> | <catches> <catch clause>
    <catch clause> ::= catch ( <formal parameter> ) <block>
    <finally > ::= finally <block>
```

FIGURE 1 – Extrait de la grammaire de Java.

3 IDE Intégré pour *draw*$^{++}$

Votre éditeur devrait permettre à un programmeur de :

— Créer un nouveau fichier de code *draw*$^{++}$.

— Ouvrir un fichier.

— Sauvegarder un fichier.

— Permettre d'ouvrir plusieurs fichiers et de basculer d'un fichier à un autre.

— Modifier un fichier.

— Exécuter un fichier.

— Souligner en rouge la première instruction erronée et proposer des corrections automatiques en vert.

— Sélectionner une partie d'un dessin.

— Zoomer, déplacer, supprimer, faire une rotation d'une partie d'un dessin déjà sélectionnée.

— Annuler la sélection d'un dessin.

— Prévoir le débogage des programmes *draw*$^{++}$.

4 Code intermédiaire

Il vous ai demandé de concevoir et d'implémenter un compilateur, en utilisant C, pour *draw*$^{++}$. Le compilateur signale en rouge la première erreur de votre code s'il est erroné et fait une proposition afin de corriger l'instruction erronée. Dans le cas contraire, un code intermédiaire, écrit en Python, est généré. Celui-ci est une forme de code exécutable de votre code (voir Figure 2).

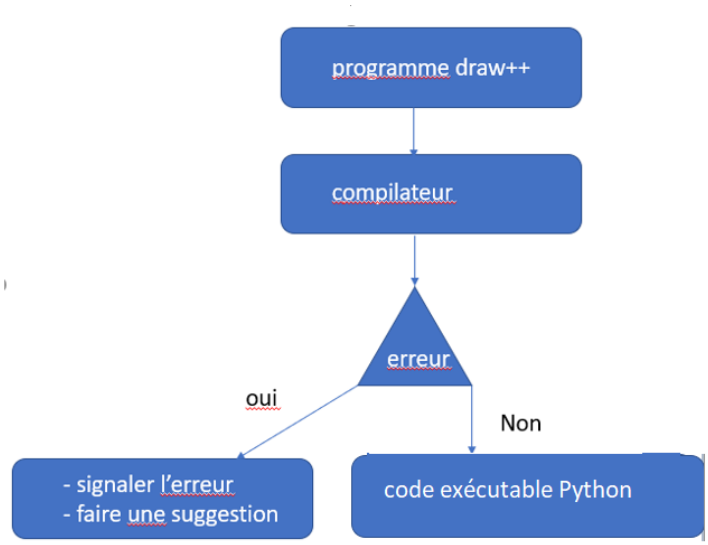

FIGURE 2 – Compilation d'un programme *draw*$^{++}$

5 Code de l'application

Taille des équipes Ce projet est un travail d'équipe. Seules les équipes de 5 ou 4 personnes sont autorisées. Désignez un interlocateur de votre groupe.

Documentation Créer une documentation complète et interactive pour aider les utilisateurs à apprendre et à utiliser le langage.

Dépôt de code Votre code doit être bien structuré, clair et bien commenté. Les commentaires

doivent être en anglais. Aucun framework pour les parseurs des grammaires ne sera autorisé. L'objectif de ce projet est de développer vos compétences en programmation et conception. Vous devrez déposer la totalité des fichiers de votre projet sur un dépôt central Git. Il en existe plusieurs disponibles gratuitement sur sites web comme github.com ou gitlab.com.

Rapport Un rapport d'une vingtaine de pages doit être rédigé en latex. Une description de la grammaire de $draw^{++}$ est exigée et des algorithmes d'analyse (parseurs lecical, syntaxique et sémantique) des instructions.

Évaluation Une évaluation individuelle de chaque élève est programmée une semaine avant les vacances de Noël. À titre indicatif, la grille d'évaluation peut être décrite comme suit :

- Code du projet, travail de groupe (30%),
- Travail individuel (50%),
- Soutenance (20 %).