Projet Compilateur

Document de conception

H4214

BOUZAABIA Mohamed - BRONNERT Mathilde

COLARD Thibault - EL GHISSASSI Ghita

GRACIA Yohan - RAHOUI Ayoub

ZERHOUDI Saber

Sommaire

Introduction

Le “Projet Compilateur”, comme son nom l’indique, est un projet dans lequel notre équipe, l’hexanome 4214, doit implémenter un compilateur. Ce compilateur fonctionne pour un sous-ensemble du langage C et est composé d’une partie Front-end et d’une partie Back-end. Ce rapport de mi-parcours contient toutes les informations nécessaires à la compréhension de notre Front-end, c’est à dire notre grammaire (construite pour un analyseur descendant), notre structure de données et toutes les fonctionnalités implémentées.

Grammaire

Voici la grammaire que nous avons utilisée pour implémenter notre compilateur.

Général

programme -> declarationVariables fonction*| ε

instruction -> ‘{‘ bloc ‘}’ |’break’ ‘;’ | structure | ‘return’ exp ‘;’ | exp ‘;’

bloc -> instruction*

Fonctions

fonction -> typeFonction NomVar ‘(‘ param ‘)’ ‘{‘ declarationVariables bloc ‘}’ | typeFonction NomVar ‘(‘ ‘)’ ‘{‘ declarationVariables bloc ‘}’

declarationVariables -> (declarationType ‘;’)*

param -> ‘void’ | typeVariable NomVar (‘,’ typeVariable NomVar)* | ε

Déclaration/Définition

declarationType -> typeVariable declaration_generale(‘,’ declaration_generale)* | typeVariable NomVar ‘[‘ Nombre ‘]’ | typeVariable NomVar ’[‘ Nombre ‘]’ ‘=’ ‘{‘ val ‘}’

val -> const (‘,’ const)*

declaration_generale -> NomVar |NomVar ‘=’ const

typeFonction -> ‘char’ | ‘int32_t’ | ‘int64_t’ | ‘void’

typeVariable -> ‘char’ | ‘int32_t’ | ‘int64_t’

const -> Nombre | CharP

Expression

exp -> exp ‘+’ exp | exp ‘*’ exp | exp ‘-’ exp | exp ‘/’ exp | exp ‘%’ exp | ‘(‘ exp ‘)’ | exp ‘<’ exp | exp ‘>’ exp | ‘!’ exp |’-’ exp | exp ‘+=’ exp |exp ‘-=’ exp | exp ‘*=’ exp |exp ‘/=’ exp | exp ‘%=’ exp |exp ‘^=’ exp | exp ‘&=’ exp|exp ‘|=’ exp |’~’ exp | exp ’==’ exp | exp ‘!=’ exp | exp ‘<=’ exp |exp ‘>=’ exp | exp ‘&&’ exp |exp ’||’ exp | exp ‘&’ exp | exp ’|’ exp | exp ’^’ exp | exp ’<<’ exp | exp ’>>’ exp | lvalue ‘++’ | lvalue ‘--’ | ‘++’ lvalue | ‘--’ lvalue | lvalue ‘=’ exp

lvalue -> NomVar | NomVar ‘[‘ exp ‘]’

exp -> Nombre | Char | NomVar ‘(‘ exp (‘,’exp)* ‘)’| lValue | ‘putchar’ ‘(‘ exp ‘)’ | ‘getchar’ ‘(‘ ‘)’

Structures de contrôle

structure -> if_statement | while_statement

IF/ELSE :

if_statement-> ‘if’ ‘(‘ exp ’)’ instruction | ‘if’ ‘(‘ exp ’)’ instruction ‘else’ instruction

WHILE :

while_statement -> ‘while’ ‘(‘ exp ‘)’ instruction

Expressions Regex

Nous avons ensuite défini les expressions régulières des terminaux suivants :

NomVar : [a-zA-Z][a-zA-Z0-9]*

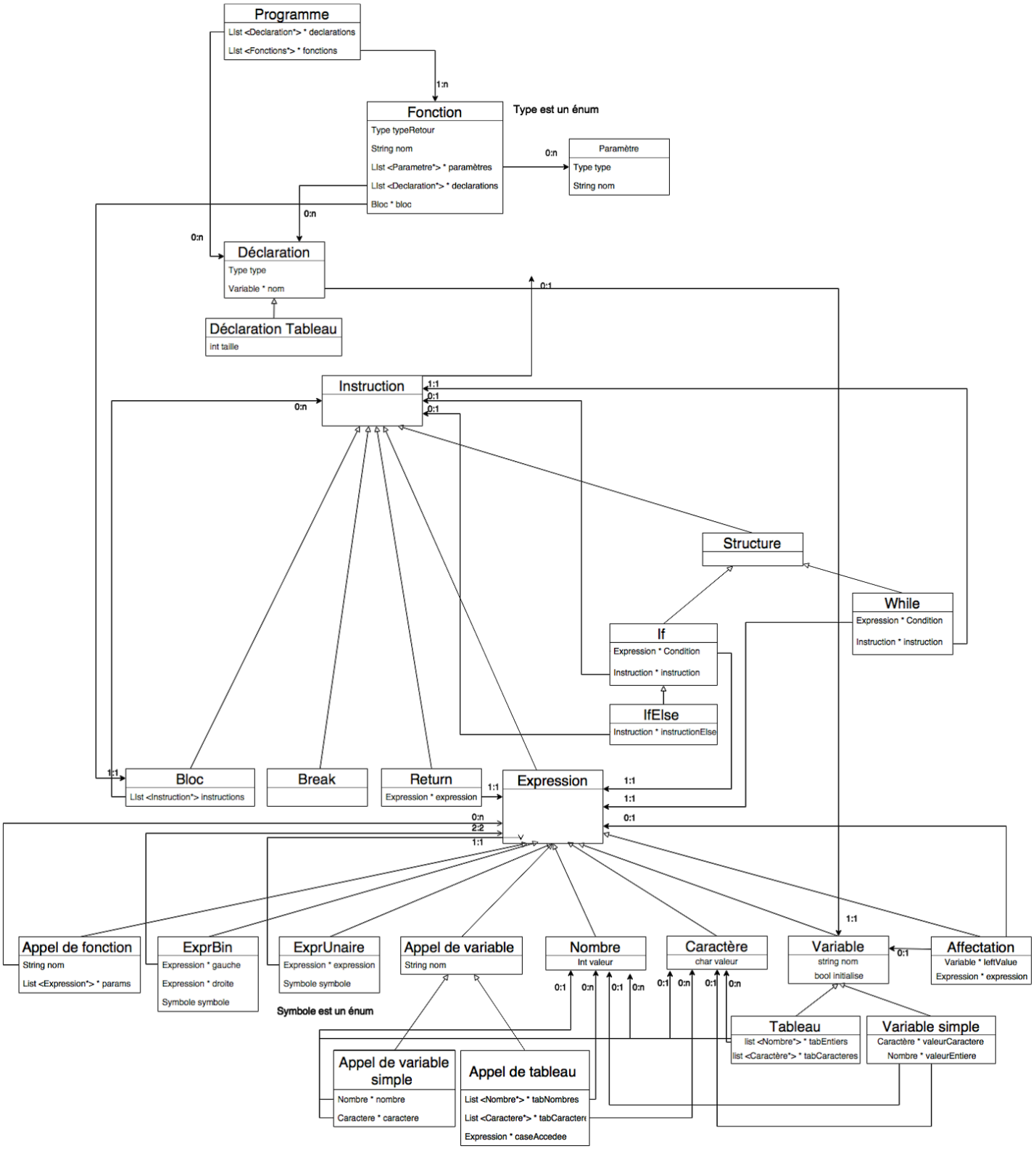
Char : '\'' ((~'\'')|'\\'~'\'') '\''

Nombre : [0-9]+

Il est à noter que nous avons décidé d’ignorer, à l’aide d’une règle “skip”, les commentaires, les importations (#) ainsi que les caractères suivants : [\t\r\n]+.

Structures de données

Voici le diagramme de classe représentant la structure de nos données :



Liste des fonctionnalités implémentées

Voici la liste des fonctionnalités que nous avons implémentées pour l'instant :

- Analyse lexicale du code à compiler.
- Analyse syntaxique du code à compiler.
- Création de la représentation en mémoire du code à compiler.
- Résolution statique des variables : si on utilise une variable non déclarée, une erreur sera produite.

Voici le sous-ensemble du langage C qui est reconnu par notre compilateur :

- Uniquement les types char, int32_t et int64_t.
- Tableaux à une dimension.
- Initialisation d'une variable possible lors de sa déclaration.
- Déclaration multiple de variables.
- Déclaration de variables globales en début de programme.
- Déclaration et définition de fonctions (type retour void possible)
- Structures de contrôle if, else et while.
- Structure de bloc avec { et }.
- Tous les opérateurs du langage C, y compris l'affectation.
- Déclarations de variables uniquement en début de fonction.
- Les fonctions putchar et getchar comme entrées-sorties.
- Les directives commençant par # en début de programme sont acceptées mais ignorées.