Implementação de Compilador 1 (IC1)

Eduardo Estefano Becker 15204477
Gustavo Henrique de Lima 15200922
Roberto Rivelino Ventura da Silva 15200945
Curso Bacharelado em Sistemas de Informação
Disciplina Introdução a Compiladores
Universidade Federal de Santa Catarina

Florianópolis, 22 de Março de 2019

1. Relatório

1.1 Papel dos integrantes

1.1.1 Artur Silva Muniz Junior (matr. 16101095)

Responsável pela implementação e testes das classes methoddecl e classdecl.

1.1.2 Eduardo Estefano Becker (matr. 1520447)

Responsável pela implementação e testes das classes vardecl e paramlist.

1.1.3 Gustavo Henrique de Lima (matr. 15200922)

Responsável pela implementação e testes das classes vardecl e paramlist.

1.1.4 Roberto Rivelino Ventura da Silva (matr. 15200945)

Responsável pela implementação e testes vardecl, methoddecl, paramlist, classbody e classdecl. Realizou também a criação e modelagem do relatório.

1.2 Analisador Sintático

Para o desenvolvimento da segunda etapa da linguagem X+++ , foi realizado a implementação e testes de um analisador sintático para classes, métodos e variáveis, assim como suas devidas expressões e parâmetros. Para isso, utilizamos os TOKENS implementados na entrega anterior.

1.2.1 Vardecl

Adicionamos a possibilidade das variáveis poderem começar com os identificadores **PUBLIC, PROTECTED, FINAL, PRIVATE**. Também adicionamos os tipos **BYTE, SHORT, LONG e FLOAT** que seguem os identificadores nas assinaturas.

```
void vardecl(RecoverySet g) throws ParseEOFException :
{
}
{
try {
   [<PUBLIC> | <PRIVATE> | <PROTECTED>] [<FINAL>] (<BYTE> | <SHORT> |< LONG> | <FLOAT> | <INT> |  <STRING> | <IDENT> )
   <IDENT> ( <LBRACKET> <RBRACKET>)* (<COMMA> <IDENT> ( <LBRACKET> <RBRACKET>)* )*
```

Note que os identificadores estão dentro de colchetes **[ ]**, tornando esses identificadores opcionais, assim como na linguagem Java.

1.2.3 Methoddcl

Para a declaração de métodos, também acrescentamos os identificadores **PUBLIC, PROTECTED, FINAL, PRIVATE** e adicionamos os tipos **BYTE, SHORT, LONG e FLOAT** seguindo os identificadores.

```
void methoddecl(RecoverySet g) throws ParseEOFException :
{
}
{
try {
   [<PUBLIC> | <PRIVATE> | <PROTECTED>] [<FINAL>] (<BYTE> | <SHORT> |< LONG> | <FLOAT> | <INT> |  <STRING> | <IDENT> ) (<LBRACKET> <RBRACKET>)*
   <IDENT> methodbody(g)
}
```

1.2.4 Classdecl

Na declaração de classe também adicionamos os identificadores **PUBLIC, PROTECTED, FINAL, PRIVATE**.

```
void classdecl(RecoverySet g) throws ParseEOFException :
{
}
{
try {
   [<PUBLIC> | <PRIVATE> | <PROTECTED>] [<FINAL>] <CLASS>  <IDENT> [ <EXTENDS> <IDENT> ] classbody(g)
}
```

1.2.5 Methodbody

Em Methodbody, modificamos o método **paramlist,** método utilizado para analisar a sintaxe do corpo do método.

```
void paramlist(RecoverySet g) throws ParseEOFException :
{
}
{
try {
   [
    (<BYTE> | <SHORT> |< LONG> | <FLOAT> | <INT> |  <STRING> | <IDENT> ) <IDENT> (<LBRACKET> <RBRACKET>)*
    (<COMMA> (<BYTE> | <SHORT> |< LONG> | <FLOAT> | <INT> |  <STRING> | <IDENT> ) <IDENT> (<LBRACKET> <RBRACKET>)* )*
   ]
}
```

Adicionamos nesse método os tipos **BYTE, SHORT, LONG e FLOAT.**

1.2.6 Classbody

Em classbody adicionamos dois LOOKAHEAD's, criando assim a possibilidade de podermos declarar variáveis fora dos métodos.

```
void classbody(RecoverySet g) throws ParseEOFException :
{
RecoverySet f2 = new RecoverySet(SEMICOLON).union(g).remove(IDENT),
            f3 = First.methoddecl.union(g).remove(IDENT),
            f4 = First.constructdecl.union(f3).remove(IDENT),
            f5 = First.vardecl.union(f4).remove(IDENT);
}
{
try {
    <LBRACE>
       [LOOKAHEAD(3)classlist(f5)]
       (
          LOOKAHEAD(4) vardecl(f2) <SEMICOLON>    |
          constructdecl(f4)                       |
          LOOKAHEAD(4) methoddecl(f3)
       )*
    <RBRACE>
}
```