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 Eduardo Estefano Becker (matr. 1520447)

Responsável pela implementação e testes dos tokens: AND, OR, XOR e NOT (operadores lógicos) . Auxiliou também na modelagem do relatório.

1.1.2 Gustavo Henrique de Lima (matr. 15200922)

Responsável pela implementação e testes dos tokens: FINAL, PUBLIC, PRIVATE e PROTECTED (qualificadores de utilizadores). Auxílio também na modelagem do relatório.

1.1.3 Roberto Rivelino Ventura da Silva (matr. 15200945)

Responsável pela implementação e testes dos tokens: BYTE, SHORT, LONG e FLOAT (variáveis e literais) e acrescentou os tokens BOOLEAN, TRUE e FALSE necessários para realizar os testes lógicos. Realizou também a criação e modelagem do relatório.

1.2 Especificação Léxica

Para o desenvolvimento da primeira etapa da linguagem X+++ , foi realizado a implementação dos tokens especificados no enunciado do trabalho, além dos tokens BOOLEAN, TRUE e FALSE necessários para a realização dos testes dos tokens lógicos:

```
/* Operadores Lógicos */

TOKEN :
{
  < AND: "&&" >
| < OR: "||" >
| < XOR: "^" >
| < NOT: "!" >
| < BOOLEAN: "boolean" >
| < TRUE: "true" >
| < FALSE: "false" >
}
/* Variáveis e literais */

TOKEN :
{
  < BYTE: "byte" >
| < SHORT: "short" >
| < LONG: "long" >
| < FLOAT: "float" >
}

/* Qualificadores de identificado */

TOKEN :
{
  < FINAL: "final" >
| < PUBLIC: "public" >
| < PRIVATE: "private" >
| < PROTECTED: "protected" >
}
```

Para o token FLOAT e LONG funcionarem da forma na qual pensamos, foi necessário acrescentar as constantes float_constant e long_constant com suas devidas expressões regulares:

```
< float_constant: ( // números flutuantes

            (<int_constant> "." <int_constant>)
) >
|
< long_constant: ( // números long

            (<int_constant> "L")
) >

```

Note que para a resolução da expressão regular, utilizamos a constante <int_constant> já existente no exemplo do capítulo 3 do livro de DELAMARO (Como Construir um Compilador: utilizando ferramenta Java) .

Tendo os tokens e as constantes devidamente implementados, chegou a hora de testá-los.

1.3 Testes dos Tokens

Nesta etapa, realizamos tanto testes que englobam os tokens adicionados para a versão X+++ quanto os tokens já presentes no exemplo do livro de DELAMARO (Linguagem X++). Também realizamos testes que demonstram alguns erros léxicos na linguagem X+++.

1.3.1 Testes sem erros léxicos

Na tabela abaixo podemos ver o código realizado para os testes sem erros léxicos da linguagem X+++ e o reconhecimento de seus tokens. O arquivo com o código fonte e os logs podem ser encontrados no arquivo zipado junto a este relatório:

Código Fonte	Tokens reconhecidos

protected testeByte() {
 	
byte variavelByte1 = 1;
byte variavelByte2 = 2;
 	
if(variavelByte1 > variavelByte2) {
print(variavelByte1);
}
else {}
}
	Line: 4 Column: 9 protected protected 37
	Line: 4 Column: 19 testeByte <IDENT> 43
	Line: 4 Column: 28 ((46
	Line: 4 Column: 29)) 47
	Line: 4 Column: 31 { { 48
	Line: 6 Column: 17 byte byte 30
	Line: 6 Column: 22 variavelByte1 <IDENT> 43
	Line: 6 Column: 36 = = 55
	Line: 6 Column: 38 3 <int_constant> 38
	Line: 6 Column: 39 ; ; 52
	Line: 8 Column: 25 if if 18
	Line: 8 Column: 27 ((46
	Line: 8 Column: 28 variavelByte1 <IDENT> 43
	Line: 8 Column: 42 > > 56
	Line: 8 Column: 44 2 <int_constant> 38
	Line: 8 Column: 45)) 47
	Line: 8 Column: 47 { { 48
	Line: 9 Column: 33 print print 21
	Line: 9 Column: 38 ((46
	Line: 9 Column: 39 variavelByte1 <IDENT> 43
	Line: 9 Column: 52)) 47

	Line: 9 Column: 53 ; ; 52 Line: 10 Column: 25 } } 49 Line: 11 Column: 33 else else 15 Line: 11 Column: 38 { { 48 Line: 13 Column: 33 } } 49 Line: 14 Column: 9 } } 49

private testeShort() {
 	
short variavelShort1 = 6;
 	
if(variavelShort1 == 6) {
 	
print(variavelShort1);
}
else {}
}
	Line: 14 Column: 9 private private 36
	Line: 14 Column: 17 testeShort <IDENT> 43
	Line: 14 Column: 27 ((46
	Line: 14 Column: 28)) 47
	Line: 14 Column: 30 { { 48
	Line: 16 Column: 17 short short 31
	Line: 16 Column: 23 variavelShort1 <IDENT> 43
	Line: 16 Column: 38 = = 55
	Line: 16 Column: 40 6 <int_constant> 38
	Line: 16 Column: 41 ; ; 52
	Line: 18 Column: 25 if if 18
	Line: 18 Column: 27 ((46
	Line: 18 Column: 28 variavelShort1 <IDENT> 43
	Line: 18 Column: 43 == == 58
	Line: 18 Column: 46 6 <int_constant> 38
	Line: 18 Column: 47)) 47
	Line: 18 Column: 49 { { 48
	Line: 19 Column: 33 print print 21
	Line: 19 Column: 38 ((46
	Line: 19 Column: 39 variavelShort1 <IDENT> 43
	Line: 19 Column: 53)) 47
	Line: 19 Column: 54 ; ; 52
	Line: 20 Column: 25 } } 49
	Line: 21 Column: 33 else else 15
	Line: 21 Column: 38 { { 48
	Line: 21 Column: 39 } } 49
	Line: 22 Column: 9 } } 49

final testeLong() {
 	
long variavelLong1 = 11837891L;
 	
if (variavelLong1 != 18837891L) {
print(variavelLong1);
}
else {}
}
	Line: 24 Column: 9 final final 34
	Line: 24 Column: 15 testeLong <IDENT> 43
	Line: 24 Column: 24 ((46
	Line: 24 Column: 25)) 47
	Line: 24 Column: 27 { { 48
	Line: 26 Column: 17 long long 32
	Line: 26 Column: 22 variavelLong1 <IDENT> 43
	Line: 26 Column: 36 = = 55
	Line: 26 Column: 38 11837891L <long_constant> 42

	Line: 26 Column: 47 ; ; 52 Line: 28 Column: 17 if if 18 Line: 28 Column: 20 ((46 Line: 28 Column: 21 variavelLong1 <IDENT> 43 Line: 28 Column: 35 != != 61 Line: 28 Column: 38 18837891L <long_constant> 42 Line: 28 Column: 48)) 47 Line: 28 Column: 50 { { 48 Line: 29 Column: 25 print print 21 Line: 29 Column: 30 ((46 Line: 29 Column: 31 variavelLong1 <IDENT> 43 Line: 29 Column: 44)) 47 Line: 29 Column: 45 ; ; 52 Line: 30 Column: 17 } } 49 Line: 31 Column: 17 else else 15 Line: 31 Column: 22 { { 48 Line: 31 Column: 23 } } 49 Line: 32 Column: 9 } } 49

```
public testeFloat() {

        float variavelFloat1 = 11.83;

                if(variavelFloat1 < 18.88 ) {
                        print(variavelFloat1);
                }
                        else {}
}
```

```
Line: 34 Column: 9 public  public  35
Line: 34 Column: 16 testeFloat  <IDENT>  43
Line: 34 Column: 26 (  (  46
Line: 34 Column: 27 )  )  47
Line: 34 Column: 29 {  {  48
Line: 36 Column: 17 float  float  33
Line: 36 Column: 23 variavelFloat1  <IDENT>  43
Line: 36 Column: 38 =  =  55
Line: 36 Column: 40 11.83  <float_constant>  41
Line: 36 Column: 45 ;  ;  52
Line: 38 Column: 25 if  if  18
Line: 38 Column: 27 (  (  46
Line: 38 Column: 28 variavelFloat1  <IDENT>  43
Line: 38 Column: 43 <  <  57
Line: 38 Column: 45 18.88  <float_constant>  41
Line: 38 Column: 51 )  )  47
Line: 38 Column: 53 {  {  48
Line: 39 Column: 33 print  print  21
Line: 39 Column: 38 (  (  46
Line: 39 Column: 39 variavelFloat1  <IDENT>  43
Line: 39 Column: 53 )  )  47
Line: 39 Column: 54 ;  ;  52
Line: 40 Column: 25 }  }  49
Line: 41 Column: 33 else  else  15
Line: 41 Column: 38 {  {  48
Line: 41 Column: 39 }  }  49
Line: 42 Column: 9 }  }  49
```

```
        public testeOperadoresLogicos() {

        boolean a = true;
        boolean b = false;

        print("a && b = " + (a&&b)); // teste
AND

        print("a || b = " + (a||b) ); // teste OR

        print("!(a && b) = " + !(a && b)); //
teste NOT

        print("a ^ b =" + (a^b)); // teste XOR
        }

}
```

```
Line: 44 Column: 9 public  public  35
Line: 44 Column: 16 testeOperadoresLogicos
<IDENT>  43
Line: 44 Column: 38 (  (  46
Line: 44 Column: 39 )  )  47
Line: 44 Column: 41 {  {  48
Line: 46 Column: 15 boolean  <IDENT>  43
Line: 46 Column: 23 a  <IDENT>  43
Line: 46 Column: 25 =  =  55
Line: 46 Column: 27 true  <IDENT>  43
Line: 46 Column: 31 ;  ;  52
Line: 47 Column: 15 boolean  <IDENT>  43
Line: 47 Column: 23 b  <IDENT>  43
Line: 47 Column: 25 =  =  55
Line: 47 Column: 27 false  <IDENT>  43
Line: 47 Column: 32 ;  ;  52
Line: 49 Column: 15 print  print  21
Line: 49 Column: 20 (  (  46
Line: 49 Column: 21 a  <IDENT>  43
Line: 49 Column: 22 &&  &&  26
Line: 49 Column: 24 b  <IDENT>  43
Line: 49 Column: 25 )  )  47
Line: 49 Column: 26 ;  ;  52
Line: 50 Column: 15 print  print  21
Line: 50 Column: 20 (  (  46
Line: 50 Column: 21 a  <IDENT>  43
Line: 50 Column: 22 ||  ||  27
Line: 50 Column: 24 b  <IDENT>  43
Line: 50 Column: 25 )  )  47
Line: 50 Column: 26 ;  ;  52
Line: 51 Column: 15 print  print  21
Line: 51 Column: 20 (  (  46
Line: 51 Column: 21 !  !  29
Line: 51 Column: 22 (  (  46
Line: 51 Column: 23 a  <IDENT>  43
Line: 51 Column: 24 &&  &&  26
Line: 51 Column: 26 b  <IDENT>  43
Line: 51 Column: 27 )  )  47
Line: 51 Column: 28 )  )  47
Line: 51 Column: 29 ;  ;  52
Line: 52 Column: 15 print  print  21
Line: 52 Column: 20 (  (  46
Line: 52 Column: 21 a  <IDENT>  43
Line: 52 Column: 22 ^  ^  28
Line: 52 Column: 23 b  <IDENT>  43
Line: 52 Column: 24 )  )  47
Line: 52 Column: 25 ;  ;  52
Line: 53 Column: 12 }  }  49
```

	Line: 55 Column: 1 } } 49 Line: 59 Column: 1 <EOF> 0

1.3.2 Testes com erros léxicos

Para a realização destes testes colocamos como entrada cadeias qua não são reconhecidas como tokens válidos, como por exemplo “3@//2” ao invés de 3002, “fi” ao invés de if, 6;8 ao invés de 6.8 ou até mesmo palavras reservadas escritas erradas como “protect” ao invés de protected ou “privati” ao invés de private:

public class testeCompila {
 	
protect testeByte() {
 	
byte variavelByte1 = 3@//2;
 	
fi(variavelByte1 > 9) {
print(variavelByte1);
}
	Line: 2 Column: 1 public public 38
	Line: 2 Column: 8 class class 13
	Line: 2 Column: 14 testeCompila <IDENT> 46
	Line: 2 Column: 27 { { 51
	Line: 4 Column: 9 protect <IDENT> 46
	Line: 4 Column: 17 testeByte <IDENT> 46
	Line: 4 Column: 26 ((49
	Line: 4 Column: 27)) 50
	Line: 4 Column: 29 { { 51
	Line: 6 Column: 17 byte byte 30
	Line: 6 Column: 22 variavelByte1 <IDENT> 46
	Line: 6 Column: 36 = = 58
	Line: 6 Column: 38 3 <int_constant> 41
	Line 6 - Invalid string found: @
	Line: 6 Column: 39 @
	<INVALID_LEXICAL> 46
	Line: 8 Column: 27 ((49
	Line: 8 Column: 28 variavelByte1 <IDENT> 46
	Line: 8 Column: 42 > > 59
	Line: 8 Column: 44 9 <int_constant> 41
	Line: 8 Column: 45)) 50
	Line: 8 Column: 47 { { 51
	Line: 9 Column: 33 print print 21
	Line: 9 Column: 38 ((49
	Line: 9 Column: 39 variavelByte1 <IDENT> 46
	Line: 9 Column: 52)) 50
	Line: 9 Column: 53 ; ; 55
	Line: 11 Column: 9 } } 52

privati testeShort() {
 	
short variavelShort1 = 0,,00085;
 	
if(variavelShort1 != 6;8) {
print(variavelShort1);
}
	Line: 13 Column: 9 privati <IDENT> 46
	Line: 13 Column: 17 testeShort <IDENT> 46
	Line: 13 Column: 27 ((49
	Line: 13 Column: 28)) 50
	Line: 13 Column: 30 { { 51
	Line: 15 Column: 17 short short 31
	Line: 15 Column: 23 variavelShort1 <IDENT> 46
	Line: 15 Column: 38 = = 58
	Line: 15 Column: 40 0 <int_constant> 41
	Line: 15 Column: 41 , , 56
	Line: 15 Column: 42 , , 56
	Line: 15 Column: 43 00085 <int_constant> 41
	Line: 15 Column: 48 ; ; 55
	Line: 17 Column: 25 if if 18
	Line: 17 Column: 27 ((49
	Line: 17 Column: 28 variavelShort1 <IDENT> 46
	Line: 17 Column: 43 != != 64
	Line: 17 Column: 46 6 <int_constant> 41
	Line: 17 Column: 47 ; ; 55
	Line: 17 Column: 48 8 <int_constant> 41
	Line: 17 Column: 49)) 50
	Line: 17 Column: 51 { { 51
	Line: 18 Column: 33 print print 21
	Line: 18 Column: 38 ((49
	Line: 18 Column: 39 variavelShort1 <IDENT> 46
	Line: 18 Column: 53)) 50
	Line: 18 Column: 54 ; ; 55
	Line: 19 Column: 9 } } 52

final testeLong() {
 	
long variavelLong1 = 11837long;
 	
if (variavelLong1 != 18837891L) {
print(variavelLong1);
 	
}
	Line: 21 Column: 9 final final 37
	Line: 21 Column: 15 testeLong <IDENT> 46
	Line: 21 Column: 24 ((49
	Line: 21 Column: 25)) 50
	Line: 21 Column: 27 { { 51
	Line: 23 Column: 17 long long 32
	Line: 23 Column: 22 variavelLong1 <IDENT> 46
	Line: 23 Column: 36 = = 58
	Line: 23 Column: 38 11837 <int_constant> 41
	Line: 23 Column: 43 long long 32
	Line: 23 Column: 47 ; ; 55
	Line: 25 Column: 17 if if 18

	Line: 25 Column: 20 ((49 Line: 25 Column: 21 variavelLong1 <IDENT> 46 Line: 25 Column: 35 != != 64 Line: 25 Column: 38 18837891L <long_constant> 45 Line: 25 Column: 48)) 50 Line: 25 Column: 50 { { 51 Line: 26 Column: 25 print print 21 Line: 26 Column: 30 ((49 Line: 26 Column: 31 variavelLong1 <IDENT> 46 Line: 26 Column: 44)) 50 Line: 26 Column: 45 ; ; 55 Line: 28 Column: 9 } } 52

public testeFloat() {
 	
float variavelFloat1 = 11,83;
 	
if(variavelFloat1 < 18;88) {
print(variavelFloat1);
}
	Line: 30 Column: 9 public public 38
	Line: 30 Column: 16 testeFloat <IDENT> 46
	Line: 30 Column: 26 ((49
	Line: 30 Column: 27)) 50
	Line: 30 Column: 29 { { 51
	Line: 32 Column: 17 float float 33
	Line: 32 Column: 23 variavelFloat1 <IDENT> 46
	Line: 32 Column: 38 = = 58
	Line: 32 Column: 40 11 <int_constant> 41
	Line: 32 Column: 42 , , 56
	Line: 32 Column: 43 83 <int_constant> 41
	Line: 32 Column: 45 ; ; 55
	Line: 34 Column: 25 if if 18
	Line: 34 Column: 27 ((49
	Line: 34 Column: 28 variavelFloat1 <IDENT> 46
	Line: 34 Column: 43 < < 60
	Line: 34 Column: 45 18 <int_constant> 41
	Line: 34 Column: 47 ; ; 55
	Line: 34 Column: 48 88 <int_constant> 41
	Line: 34 Column: 51)) 50
	Line: 34 Column: 53 { { 51
	Line: 35 Column: 33 print print 21
	Line: 35 Column: 38 ((49
	Line: 35 Column: 39 variavelFloat1 <IDENT> 46
	Line: 35 Column: 53)) 50
	Line: 35 Column: 54 ; ; 55
	Line: 36 Column: 9 } } 52

public testeOperadoresLogicos() {
 	
boolean a = true;
boolean b = false;
 	
print(a&&&b);
print(a|||b);
print(!!(a&&b));
print(a^^b);
}
 	
}
	Line: 38 Column: 9 public public 38
	Line: 38 Column: 16
	testeOperadoresLogicos <IDENT> 46
	Line: 38 Column: 38 ((49
	Line: 38 Column: 39)) 50
	Line: 38 Column: 41 { { 51
	Line: 40 Column: 15 boolean boolean 34
	Line: 40 Column: 23 a <IDENT> 46
	Line: 40 Column: 25 = = 58
	Line: 40 Column: 27 true true 35
	Line: 40 Column: 31 ; ; 55
	Line: 41 Column: 15 boolean boolean 34
	Line: 41 Column: 23 b <IDENT> 46
	Line: 41 Column: 25 = = 58
	Line: 41 Column: 27 false false 36
	Line: 41 Column: 32 ; ; 55
	Line: 43 Column: 15 print print 21
	Line: 43 Column: 20 ((49
	Line: 43 Column: 21 a <IDENT> 46
	Line 43 - Invalid string found: &&&
	Line: 43 Column: 22 &&&
	<INVALID_LEXICAL> 46
	Line: 43 Column: 26)) 50
	Line: 43 Column: 27 ; ; 55
	Line: 44 Column: 15 print print 21
	Line: 44 Column: 20 ((49
	Line: 44 Column: 21 a <IDENT> 46
	Line 44 - Invalid string found: |||
	Line: 44 Column: 22 |||
	<INVALID_LEXICAL> 46
	Line: 44 Column: 26)) 50
	Line: 44 Column: 27 ; ; 55
	Line: 45 Column: 15 print print 21
	Line: 45 Column: 20 ((49
	Line: 45 Column: 21 ! ! 29
	Line: 45 Column: 22 ! ! 29
	Line: 45 Column: 23 ((49
	Line: 45 Column: 24 a <IDENT> 46
	Line: 45 Column: 25 && && 26
	Line: 45 Column: 27 b <IDENT> 46
	Line: 45 Column: 28)) 50
	Line: 45 Column: 29)) 50
	Line: 45 Column: 30 ; ; 55
	Line: 46 Column: 15 print print 21
	Line: 46 Column: 20 ((49
	Line: 46 Column: 21 a <IDENT> 46
	Line 46 - Invalid string found: ^^
	Line: 46 Column: 22 ^^

	<INVALID_LEXICAL> 46 Line: 46 Column: 25)) 50 Line: 46 Column: 26 ; ; 55 Line: 47 Column: 12 } } 52 Line: 49 Column: 1 } } 52 Line: 50 Column: 1 <EOF> 0 4 Lexical Errors found

Podemos observar que na linha 23, coluna 38 o compilador não consegue identificar a entrada “11837long” como uma <long_constant> e a divide entre uma <int_constant> : 11837 e uma palavra restrita “long”. Isso é considerável se levarmos em consideração que a constante do token para long possui o padrão <int_constant> “L” (constante inteira seguida de “L” maiúsculo). Seguindo os logs, encontramos outros erros léxicos, como a utilização de “&&&” ao invés do padrão “&&” ou “|||” ao invés do padrão “||”.