Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Compiladores I - DCC053
Trabalho Prático 2
Um Front-End Completo do Compilador de SmallL

Hugo Araújo de Sousa
(2013007463)

2º Semestre de 2017

Conteúdo

1 Descrição do Problema 2
2 Metodologia 2
2.1 Análise Léxica 3
2.2 Análise Sintática 3
2.3 Análise Semântica 3
2.4 Geração de Código Intermediário 3
3 Código Fonte 3
4 Compilação e Uso 3
5 Testes e Resultados 4
5.1 right1.txt 4
5.2 right2.txt 4
5.3 right3.txt 6
5.4 wrong1.txt 7
5.5 wrong2.txt 8
5.6 wrong3.txt 9
5.7 wrong4.txt 9
6 Conclusão 10
7 Referências 11

1 Descrição do Problema

Em projetos de construção de compiladores, dizemos que o **front-end** do compilador é a parte do mesmo responsável pela análise do código fonte. Sendo assim, essa é a parte dependente da linguagem fonte. Ela consiste geralmente de três etapas:

1. Análise Léxica: Lê o o programa fonte agrupando os símbolos que o compõem em sequências significativas. Para cada uma dessas sequências, armazena-se informações importantes para etapas posteriores.

2. Análise Sintática: Utiliza a saída da análise léxica para impor uma estrutura gramatical ao programa fonte.

3. Análise Semântica: Verifica a consistência semântica do programa fonte, estabelecida na própria definição da linguagem. Nessa etapa ocorre a verificação de tipos.

4. Geração de Código Intermediário: Gera uma representação intermediária de baixo nível ou do tipo linguagem de máquina do programa fonte. Essa representação deve ser facilmente traduzida para a máquina alvo.

Nesse trabalho, é construído um front-end para a mini-linguagem SmallL, definida pela gramática mostrada a seguir.

```
1  program               ->       block
2  block                 ->       { decls stmts }
3  decls                 ->       decls decl | e
4  decl                  ->       type id ;
5  type                  ->        type [ num ] | basic
6  stmts                 ->       stmts stmt | e
7  stmt                   ->        loc = bool ;
8                           |       if ( bool ) stmt
9                           |       if ( bool ) stmt else stmt
10                          |       while ( bool ) stmt
11                          |       do stmt while ( bool ) ;
12                          |        break ;
13                          |       block
14 loc                   ->       loc [ bool ] | id
15 bool                   ->        bool || join | join
16 join                  ->        join && equality | equality
17 equality         ->        equality == rel | equality != rel | rel
18 rel                   ->        expr < expr | expr <= expr | expr >= expr | expr > expr | expr
19 expr                  ->        expr + term | expr - term | term
20 term                   ->       term * unary | term / unary | unary
21 unary                   ->        ! unary | - unary | factor
22 factor                   ->        ( bool ) | loc | num | real | true | false
```

Gramática da linguagem SmallL. 'e' indica a string vazia.

2 Metodologia

O front-end segue a implementação do Livro Texto da disciplina. Dessa forma, o código (escrito na linguagem Java), foi dividido em cinco pacotes: **main**, **lexer**, **symbols**, **parser** e **inter**. Esses pacotes são descritos na Seção 3. Cada um deles representa uma etapa do front-end.

2.1 Análise Léxica

A análise léxica é feita de forma tradicional, lendo os caracteres do programa de entrada e tentando agrupá-los em tokens reconhecidos pela gramática da linguagem. Para armazenar informações de símbolos, a tabela de símbolos foi implementada utilizando-se uma tabela Hash.

O analisador léxico armazena o número da linha do programa fonte em que se encontra a fim de fornecer essa informação ao usuário no caso de erro.

2.2 Análise Sintática

O analisador sintático atua sempre em comunicação com o analisador léxico, requisitando tokens e usando-os para construir a estrutura gramatical definida pela gramática da linguagem. Para esse trabalho a análise sintática foi implementada utilizando o algoritmo de análise descendente.

2.3 Análise Semântica

A análise semântica implementada no trabalho se resume à verificação de tipos, que é então feita de forma integrada com a geração de código intermediário.

2.4 Geração de Código Intermediário

A geração de código intermediário segue o esquema de geração de código com três endereços. Esse código suporta expressões ariméticas, expressões de desvio e declarações.

3 Código Fonte

Os pacotes do projeto são descritos a seguir. Todo o código fonte pode ser obtido em um repositório do GitHub [1].

- **main** Programa principal, realiza a integração de todos os outros módulos.
- **lexer** Implementa a análise léxica do front-end.
- **parser** Analisador semântico.
- **symbols** Análise semântica, notadamente verificação de tipos.
- **inter** Geração de código intermediário de três endereços.

4 Compilação e Uso

A fim de facilitar a compilação do projeto, um arquivo Makefile é fornecido na diretório src. Para compilar o projeto, basta estar no diretório src e executar o comando **make**.

Já para executar o front-end, executa-se, na pasta src, o seguinte comando:

java main.Main < arquivo_entrada

Onde arquivo_entrada indica o nome do arquivo fonte a ser analisado.

[1] https://github.com/ha2398/compiladores1-tps/tree/master/tp2

5 Testes e Resultados

A fim de verificar o funcionamento correto do projeto, alguns testes foram desenvolvidos. Suas entradas e saídas são apresentadas a seguir.

5.1 right1.txt

Descreve um arquivo de teste simples, de sintaxe correta, que simplesmente executa algumas declarações e comandos de loop, atribuições e expressões aritméticas.

- Entrada

```
{                       // Arquivo teste
        int i; int j; float v; float x; float[100] a;
        while( true ) {
                do i = i+1; while( a[i] < v);
                do j = j-1; while( a[j] > v);
                if( i >= j) break;
                x = a[i]; a[i] = a[j]; a[j] = x;
        }
}
```

- Saída

```
------------------BEGIN INPUT------------------
{                       // Arquivo teste
        int i; int j; float v; float x; float[100] a;
        while( true ) {
                do i = i+1; while( a[i] < v);
                do j = j-1; while( a[j] > v);
                if( i >= j) break;
                x = a[i]; a[i] = a[j]; a[j] = x;
        }
}
--------------------END INPUT--------------------
L1:L3:        i = i + 1
        t1 = i * 8
        t2 = a [ t1 ]
        if t2 < v goto L3
L4:        j = j - 1
        t3 = j * 8
        t4 = a [ t3 ]
        if t4 > v goto L4
L6:        iffalse i >= j goto L8
L9:        goto L2
L8:        t5 = i * 8
        x = a [ t5 ]
L10:        t6 = i * 8
        t7 = j * 8
        t8 = a [ t7 ]
        a [ t6 ] = t8
L11:        t9 = j * 8
        a [ t9 ] = x
        goto L1
L2:
```

5.2 right2.txt

Implementação do algoritmo Insertion Sort na linguagem SmallL. A sintaxe está correta.

- Entrada

```
{
        float [8] numbers;
        float temp;
        int length;
        int j;
        int i;

        numbers[0] = 90;
        numbers[1] = -55.19;
        numbers[2] = 0;
        numbers[3] = 2592;
        numbers[4] = 1.67;
        numbers[5] = 3.1415;
        numbers[6] = 2.71;
        numbers[7] = 18;

        length = 8;

        i = 0;
        while ( i < length ) {
                j = i;

                while (j > 0 && numbers[j] < numbers[j - 1]) {
                        temp = numbers[j];
                        numbers[j] = numbers[j - 1];
                        numbers[j - 1] = temp;
                        j = j -1;
                }

                i = i + 1;
        }
}
```

- Saída

```
------------------BEGIN INPUT------------------
{
        float [8] numbers;
        float temp;
        int length;
        int j;
        int i;

        numbers[0] = 90;
        numbers[1] = -55.19;
        numbers[2] = 0;
        numbers[3] = 2592;
        numbers[4] = 1.67;
        numbers[5] = 3.1415;
        numbers[6] = 2.71;
        numbers[7] = 18;

        length = 8;

        i = 0;
        while ( i < length ) {
                j = i;

```

```
24                  while (j > 0 && numbers[j] < numbers[j - 1]) {
25                          temp = numbers[j];
26                          numbers[j] = numbers[j - 1];
27                          numbers[j - 1] = temp;
28                          j = j -1;
29                  }
30
31                  i = i + 1;
32          }
33  }
34  --------------------END INPUT--------------------
35  L1:         t1 = 0 * 8
36          numbers [ t1 ] = 90
37  L3:         t2 = 1 * 8
38          t3 = minus 55.19
39          numbers [ t2 ] = t3
40  L4:         t4 = 2 * 8
41          numbers [ t4 ] = 0
42  L5:         t5 = 3 * 8
43          numbers [ t5 ] = 2592
44  L6:         t6 = 4 * 8
45          numbers [ t6 ] = 1.6700001
46  L7:         t7 = 5 * 8
47          numbers [ t7 ] = 3.1414998
48  L8:         t8 = 6 * 8
49          numbers [ t8 ] = 2.71
50  L9:         t9 = 7 * 8
51          numbers [ t9 ] = 18
52  L10:         length = 8
53  L11:         i = 0
54  L12:         iffalse i < length goto L2
55  L13:         j = i
56  L14:         iffalse j > 0 goto L15
57          t10 = j * 8
58          t11 = numbers [ t10 ]
59          t12 = j - 1
60          t13 = t12 * 8
61          t14 = numbers [ t13 ]
62          iffalse t11 < t14 goto L15
63  L16:         t15 = j * 8
64          temp = numbers [ t15 ]
65  L17:         t16 = j * 8
66          t17 = j - 1
67          t18 = t17 * 8
68          t19 = numbers [ t18 ]
69          numbers [ t16 ] = t19
70  L18:         t20 = j - 1
71          t21 = t20 * 8
72          numbers [ t21 ] = temp
73  L19:         j = j - 1
74          goto L14
75  L15:         i = i + 1
76          goto L12
77  L2:
```

5.3 right3.txt

Calcula o décimo número da sequência de Fibonacci. A sintaxe está correta.

- Entrada

```
{
        int n; int u; int result; int i; int t;

        n = 10;
        u = 0;
        result = 1;

        i = 2;
        while (i <= n) {
                t = u + result;
                u = result;
                result = t;

                i = i + 1;
        }
}
```

- Saída

```
------------------BEGIN INPUT------------------
{
        int n; int u; int result; int i; int t;

        n = 10;
        u = 0;
        result = 1;

        i = 2;
        while (i <= n) {
                t = u + result;
                u = result;
                result = t;

                i = i + 1;
        }
}
--------------------END INPUT--------------------
L1:        n = 10
L3:        u = 0
L4:        result = 1
L5:        i = 2
L6:        iffalse i <= n goto L2
L7:        t = u + result
L8:        u = result
L9:        result = t
L10:        i = i + 1
        goto L6
L2:
```

Os arquivos de teste a seguir tratam-se dos mesmos arquivos mostrados acima, porém com pequenos erros de sintaxe.

5.4 wrong1.txt

- Entrada

```
{
        int i; int j; float v; float x; float[100) a;
        while( true ) {
```

```
4                   do i = i+1; while( a[i] < v);
5                   do j = j-1; while( a[j] > v);
6                   if( i >= j) break;
7                   x = a[i]; a[i] = a[j]; a[j] = x;
8           }
9   }
```

- Saída

```
1   -------------------BEGIN INPUT-------------------
2   {
3           int i; int j; float v; float x; float[100]Exception in thread "main"
4           java.lang.Error: near line2: syntax error
5           at parser.Parser.error(Parser.java:24)
6           at parser.Parser.match(Parser.java:31)
7           at parser.Parser.dims(Parser.java:80)
8           at parser.Parser.type(Parser.java:73)
9           at parser.Parser.decls(Parser.java:56)
10          at parser.Parser.block(Parser.java:47)
11          at parser.Parser.program(Parser.java:35)
12          at main.Main.main(Main.java:11)
```

5.5 wrong2.txt

- Entrada

```
1   {
2           int i; int j; float v; float x; float[100] a;
3           while( true ) {
4                   do i = i+1; while( a[i] < v);
5                   do j = j-1; while( a[j] > v);
6                   if( i >= j) break;
7                   x = a[i]; a[i] = a[j]; c[j] = x;
8           }
9   }
```

- Saída

```
1   -------------------BEGIN INPUT-------------------
2   {
3           int i; int j; float v; float x; float[100] a;
4           while( true ) {
5                   do i = i+1; while( a[i] < v);
6                   do j = j-1; while( a[j] > v);
7                   if( i >= j) break;
8                   x = a[i]; a[i] = a[j]; c[Exception in thread "main" java.lang.Error:
9                   near line7: c undeclared
10          at parser.Parser.error(Parser.java:24)
11          at parser.Parser.assign(Parser.java:168)
12          at parser.Parser.stmt(Parser.java:157)
13          at parser.Parser.stmts(Parser.java:92)
14          at parser.Parser.stmts(Parser.java:92)
15          at parser.Parser.stmts(Parser.java:92)
16          at parser.Parser.stmts(Parser.java:92)
17          at parser.Parser.stmts(Parser.java:92)
18          at parser.Parser.stmts(Parser.java:92)
19          at parser.Parser.block(Parser.java:48)
20          at parser.Parser.stmt(Parser.java:154)
21          at parser.Parser.stmt(Parser.java:128)
22          at parser.Parser.stmts(Parser.java:92)
```

```
23          at parser.Parser.block(Parser.java:48)
24          at parser.Parser.program(Parser.java:35)
25          at main.Main.main(Main.java:11)
```

5.6 wrong3.txt

- Entrada

```
1  {
2          int i; int j; float v; float x; float[100] a;
3          while( True ) {
4                  do i = i+1; while( a[i] < v);
5                  do j = j-1; while( a[j] > v);
6                  if( i >= j) break;
7                  x = a[i]; a[i] = a[j]; a[j] = x;
8          }
9  }
```

- Saída

```
1  ------------------BEGIN INPUT------------------
2  {
3          int i; int j; float v; float x; float[100] a;
4          while( True Exception in thread "main" java.lang.Error:
5          near line3: True undeclared
6          at parser.Parser.error(Parser.java:24)
7          at parser.Parser.factor(Parser.java:308)
8          at parser.Parser.unary(Parser.java:269)
9          at parser.Parser.term(Parser.java:249)
10         at parser.Parser.expr(Parser.java:237)
11         at parser.Parser.rel(Parser.java:220)
12         at parser.Parser.equality(Parser.java:208)
13         at parser.Parser.join(Parser.java:196)
14         at parser.Parser.bool(Parser.java:184)
15         at parser.Parser.stmt(Parser.java:126)
16         at parser.Parser.stmts(Parser.java:92)
17         at parser.Parser.block(Parser.java:48)
18         at parser.Parser.program(Parser.java:35)
19         at main.Main.main(Main.java:11)
```

5.7 wrong4.txt

- Entrada

```
1  {
2          float [8] numbers;
3          float temp;
4          int length;
5          int j;
6          int i;
7
8          numbers[0] = 90;
9          numbers[1] = -55.19;
10         numbers[2] = 0;
11         numbers[3] = 2592;
12         numbers[4] = 1.67;
13         numbers[5] = 3.1415;
14         numbers[6] = 2,1;
15         numbers[7] = 18;
16
```

```
17          length = 8;
18
19          i = 0;
20          while ( i < length ) {
21                  j = i;
22
23                  while (j > 0 && numbers[j] < numbers[j - 1]) {
24                          temp = numbers[j];
25                          numbers[j] = numbers[j - 1];
26                          numbers[j - 1] = temp;
27                          j = j -1;
28                  }
29
30                  i = i + 1;
31          }
32  }
```

- Saída

```
1   ------------------BEGIN INPUT------------------
2   {
3           float [8] numbers;
4           float temp;
5           int length;
6           int j;
7           int i;
8
9           numbers[0] = 90;
10          numbers[1] = -55.19;
11          numbers[2] = 0;
12          numbers[3] = 2592;
13          numbers[4] = 1.67;
14          numbers[5] = 3.1415;
15          numbers[6] = 2,Exception in thread "main" java.lang.Error:
16          near line14: syntax error
17          at parser.Parser.error(Parser.java:24)
18          at parser.Parser.match(Parser.java:31)
19          at parser.Parser.assign(Parser.java:179)
20          at parser.Parser.stmt(Parser.java:157)
21          at parser.Parser.stmts(Parser.java:92)
22          at parser.Parser.stmts(Parser.java:92)
23          at parser.Parser.stmts(Parser.java:92)
24          at parser.Parser.stmts(Parser.java:92)
25          at parser.Parser.stmts(Parser.java:92)
26          at parser.Parser.stmts(Parser.java:92)
27          at parser.Parser.stmts(Parser.java:92)
28          at parser.Parser.block(Parser.java:48)
29          at parser.Parser.program(Parser.java:35)
30          at main.Main.main(Main.java:11)
```

6 Conclusão

O trabalho foi desenvolvido sem dificuldades a partir do código já fornecido na especificação. Apenas algumas correções de sintaxe foram necessárias juntamente com ajustes para listagem de código fonte. De maneira geral, o trabalho em questão permitiu um maior entendimento sobre o front-end de um compilador, fornecendo uma visão mais detalhada das etapas que o compõem.

7 Referências

- Aho, A.V.; Sethi, R.; Ullman, J.D. Compilers Principles, Techniques, and Tools, Addison Wesley, 1986.