Trabalho prático compiladores - Entrega 2

Nome: **Ian Carlos Afonso da Silva**

Código fonte em anexo.

Como usar o compilador:

- Colar o arquivo na pasta src/files do projeto
- Executar o projeto pelo terminal, enviando como argumento o nome do arquivo (com o ".txt" no final.
- Caso não seja enviado, o programa carregará o arquivo identificado na classe Main.java.

Abordagem:

Foi utilizada a mesma abordagem exemplificada nos slides e no livro referência. Após a abertura do arquivo, a classe Main faz uma chamada do Lexer, e ordena que o mesmo faça a análise léxica do arquivo carregado. No início, são criadas as palavras reservadas e, logo após, inicia-se o scan do arquivo, caractere a caractere, a fim de fazer agrupamentos e retornar os tokens.

Após a análise léxica do arquivo, a lista de Tokens é enviada para o Sintaxer, que é a classe responsável pela análise sintática do programa. Após receber a lista de tokens, o módulo sintático posiciona o iterador (Token current) de tokens para o primeiro token válido. Após o posicionamento, é chamada a primeira regra da gramática (Program) e, com isso, a sequência de tokens é avaliada através das Tags de cada token. Caso a Tag não corresponda ao esperado, é lançada uma exception fatal, indicando que o token não é o esperado.

Adequação de gramática:

Com a gramática especificada, era impossível exibir uma String (literal, na linguagem do programa) para o usuário. Com isso, adicionei uma derivação da regra Writable, onde:

writable -> simple-exp | literal

Assim, é possível utilizar o comando write tanto para exibir valores de variáveis, quanto exibir textos literais.

Na mesma linha de raciocínio, não era possível fazer operações de strings. Ou seja, a gramática atual não aceita uma atribuição de string + literal. Logo, também adicionei a seguinte regra na gramática:

factor -> identifier | constant | literal | "(" expression ")"

Problemas do compilador:

Tive algumas dificuldades na hora de exibir uma mensagem de erro para o usuário. Assim sendo, erros envolvendo tokens que não são de caractere único exibem uma mensagem um pouquinho esquisita para o usuário, tais como:

```
Iniciando análise léxica do arquivo ./src/files/test5_corrigido.txt
[Syntatic Error: Expected tag 283, got ć at line 1]
```

Quando encontro algum comentário que esteja em várias linhas, o contador de linhas não está atualizando. Assim, se eu tenho um comentário de 2 linhas à partir da linha 5, um erro na linha 10 exibe o erro na linha 9, por exemplo.

Resultados de testes

- Test1

```
Iniciando análise léxica do arquivo ./src/files/test1.txt
[Syntatic Error: Expected tag 41, got Ą at line 10]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Correção do erro: corrigir a operação da linha 10.

```
10        result = (a * c)/(b + 5 - 345);
```

- Test2

```
Iniciando análise léxica do arquivo ./src/files/test2.txt
[Um comentário não foi fechado]
```

Correção do erro: fechar o comentário da linha 2.

Erro de tipagem (linha 4):

```
4     a, 9valor, b_1, b_2 : int;
```

```
Iniciando análise léxica do arquivo ./src/files/test2_1.txt
[Syntatic Error: Expected Type (int, float or string), got Word{lexema=a} at line 3]
```

Correção do erro: corrigir a declaração de variáveis de acordo com a gramática.

```
4     int a, valor, b_1, b_2;
```

Caractere inválido (linha 7):

```
Iniciando análise léxica do arquivo ./src/files/test2_1.txt
[Syntatic Error: Expected tag 61, got : at line 7]
```

Correção do erro: remover o caractere “:” inválido.

```
8     b_1 = a * a;
```

Comando Write (linha 13):

```
Iniciando análise léxica do arquivo ./src/files/test2_1.txt
[Syntatic Error: Expected tag 61, got ( at line 12]
```

Correção do erro: a linguagem é case sensitive. Logo, “write” é diferente de Write (que é considerado como Identifier)

```
13        write (b2);
```

- Test3

Comando inválido (linha 1):

```
Iniciando análise léxica do arquivo ./src/files/test3.txt
[Syntatic Error: Expected tag 283, got é at line 1]
```

```
1   classe Teste3
```

Correção do erro: o identificador de início do programa é o comando “class”, e não “classe”.

Comando inválido (linha 10):

```
Iniciando análise léxica do arquivo ./src/files/test3_1.txt
[Syntatic Error: Expected tag 61, got ( at line 9]
```

```
10    IF (qtd>=2){
```

Correção do erro: a sintaxe é case sensitive. Assim, o comando IF foi definido como variável, e aguarda uma execução de atribuição.

- Test4

Malformed Literal (linha 5) [Erro léxico]:

```
Iniciando análise léxica do arquivo ./src/files/test4.txt
[Um literal não foi fechado]
```

```
5    write("Digite o seu nome: );
```

Correção do erro: fechar a string informada no write:

```
5    write("Digite o seu nome: ");
```

Estrutura do código (linhas 1-4):

```
Iniciando análise léxica do arquivo ./src/files/test4.txt
[Syntatic Error: Expected tag 283, got { at line 1]
```

```
1    {
2    // Outro programa de teste
3    int idade, j, k, @total;
4    string nome, texto;
```

Correção do erro: a gramática da linguagem exige que o programa se inicie com **class** identifier, e só após declarar as variáveis, é necessário o caractere “{“.
Assim, teríamos:

```
1    class Test4
2    // Outro programa de teste
3        int idade, j, k, total;
4        string nome, texto;
5    {
```

- Test5

Estrutura da gramática (linha 2):

```
1    class MinhaClasse
2    { float a, b, c;
```

```
Iniciando análise léxica do arquivo ./src/files/test5.txt
[Syntatic Error: Expected Type (int, float or string), got Word{lexema={} at line 2]
```

Correção do erro: o caractere “{“ é obrigatório após a declaração das variáveis (decl-list).

Caractere faltando (linha 9):

```
9    read(c;
```

```
Iniciando análise léxica do arquivo ./src/files/test5.txt
[Syntatic Error: Expected tag 41, got ; at line 9]
```

Correção do erro: é necessário fechar o parênteses do comando read.

Caractere inválido na atribuição (linha 10):

```
10      maior := 0;
```

```
Iniciando análise léxica do arquivo ./src/files/test5.txt
[Syntatic Error: Expected tag 61, got : at line 10]
```

Correção do erro: é necessário remover o caractere inválido ":" da atribuição.

Operação inválida (linha 11):

```
11      if ( a>b && a>c )
```

```
Iniciando análise léxica do arquivo ./src/files/test5.txt
[Syntatic Error: Expected tag 41, got > at line 11]
```

Correção do erro: operações lógicas não são recursivas à esquerda ou direita. Assim, é necessário envolver as expressões entre parênteses.

```
11      if ( (a>b) && (a>c) )
```

Estrutura de if (linhas 12-20):

```
12      if ( (a>b) && (a>c) ) {
13          maior = a;
14      }else{
15          if (b>c) {
16              maior = b;
17          }else{
18              maior = c;
19          };
20      };
```

Correção do erro: é necessário informar os caracteres "{" e "}" entre os comandos if e else. Além disso, também é necessário finalizar o comando com o caractere ";"

Testes extras

- Test6 **(Média de 2 números)**

```
1    class Test6
2    int a, b, c;
3    {
4        write("Digite o primeiro número");
5        read(a);
6        write("Digite o segundo numero número");
7        read(b);
8
9        c = (a + b) /2;
10       texto = "a media dos números é "+c;
11       write(texto);
12   }
13
```

```
Iniciando análise léxica do arquivo ./src/files/test6.txt
Fim da análise sintática
```

- Test7 **(Soma de 2 números)**

```
1    class Test7
2    int a, b;
3    {
4        write("Digite o primeiro número");
5        read(a);
6        write("Digite o segundo numero número");
7        read(b);
8
9        c = a + b;
10       texto = "a soma dos números é "+c;
11       write(texto);
12   }
13
```

```
Iniciando análise léxica do arquivo ./src/files/teste7.txt
Fim da análise sintática
```