UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação

SCC0605 - Teoria da Computação e Compiladores
Profº Drº Thiago Alexandre Salgueiro Pardo

Analisador Léxico

Beatriz Aimée Teixeira Furtado Braga - Nº USP 12547934
Carlos Henrique Craveiro Aquino Veras - Nº USP 12547187
Ivan Roberto Wagner Pancheniak Filho - Nº USP 12624224
Jade Bortot de Paiva - Nº USP 11372883

São Carlos
2024

1 Introdução

O analisador léxico é a primeira etapa de um compilador, a qual executa as principais funções de ler do programa-fonte, identificar dos tokens/símbolos e relatar erros. Tais erros relacionados apenas construção dos símbolos e não ao contexto do código em PL/0.

Ele também faz o trabalho de ler comentários e caracteres não imprimíveis, a manipulação da tabela de símbolos e o relacionamento as mensagens de erro emitidas pelo compilador com o programa-fonte.

Nesse projeto, também foi construído um código em C que faza análise léxica de um determinado arquivo.

Esse código em C, os autômatos e os teste utilizados são encontrados nesse repositório do GitHub.

2 Decisões

Aqui será relatado, de forma cronológica, os passos para a construção do autômato final que faz a análise léxica da linguagem PL/0.

2.1 Tabela de caracteres reservados e de tipos

Inicialmente, ao analisar a linguagem PL/0, foi preciso construir a tabela de tipos, tabela 1, fundamental para o nosso analisador, onde faz referência a tabela outra tabela muito importante que é a tabela de caracteres reservados, tabela 2, que é fundamental para que seja possível fazer o autômato referente a linguagem e a análise léxica.

String	**Rule**
ident	(a-z + A-z)(a-z + A-Z + 0-9)*
number	(0-9)(0-9)*
reserved	see reserved table

Tabela 1: Tabela de tipos.

Symbol	Token
PROCEDURE	procedure
CONST	const
VAR	var
CAL	call
BEGIN	begin
END	end
IF	if
THEN	then
WHILE	while
DO	do
ODD	odd
+	plus
-	minus
*	mult
/	div
<>	diff
<	less
>	bigger
<=	less_eq
>=	bigg_eq
=	equal
:=	assign
.	end_prog
;	end_exp
,	separator
(	open_exp
)	close_exp

Tabela 2: Tabela de caracteres reservados.

2.2 Autômatos construídos

2.2.1 Palavras reservadas

Para a construção do autômato que faz a analise léxica para a linguagem PL/0, iniciamos construindo um autômato que faz a verificação e identificação das palavras reservadas armazenadas na tabela de palavras reservadas já criada anteriormente (indicar a tabela aqui).

Ao identificar uma palavra presente na tabela, esse autômato o retorna apropriadamente, como mostra na figura 2.2.1.

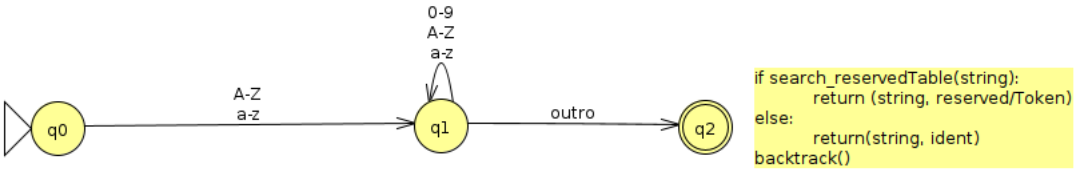

Figura 1: Primeiro rascunho do autômato de verificação e identificação de palavras reservadas.

2.2.2 Caracteres invisíveis

A partir do autômato da figura 2.2.2, foi feita a adição dos caracteres de espaço e das variações da representação do enter (\n, \r, \t) os quais chamamos de *"invisíveis_chars"*, que podem estar presentes antes de ler a palavra.

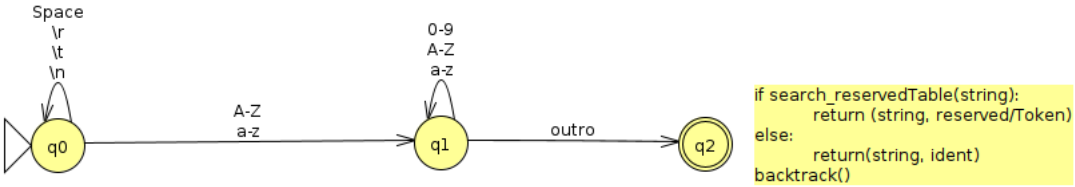

Figura 2: Adição para identificação de caracteres invisíveis.

2.2.3 Comentários

No autômato da figura 2.2.3, foi adicionado o estado q3, o qual faz o processamento de comentários de acordo com a linguagem PL/0.

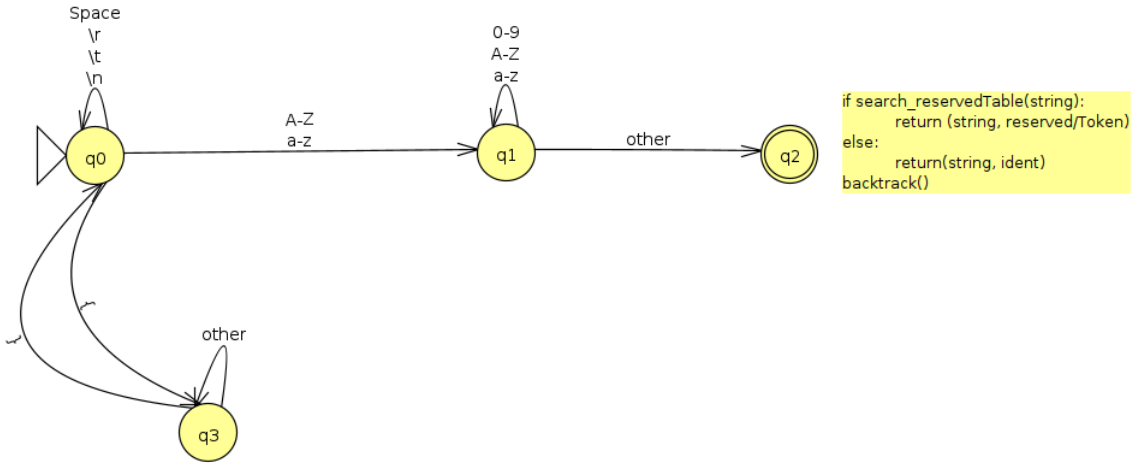

Figura 3: Adição para identificação de comentários.

2.2.4 Números

No autômato da figura 2.2.4, está sendo realizada a etapa de processamento dos números sem ou antes da inserção de letras.

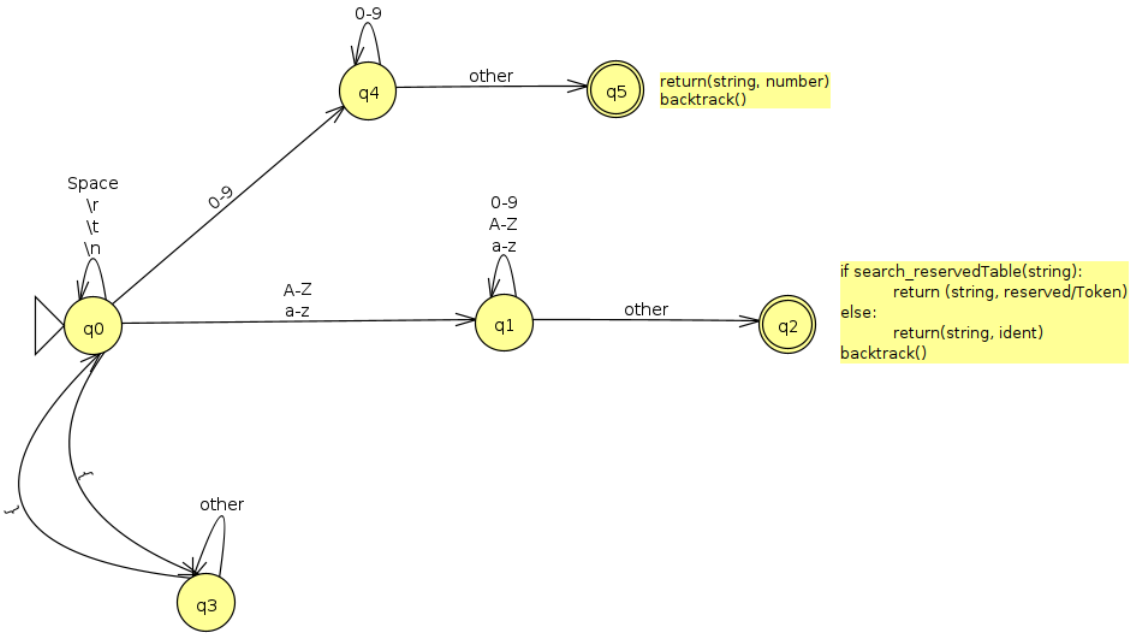

Figura 4: Adição do estado para processamento de números.

2.2.5 Símbolos reservados

No autômato da figura 2.2.5, incluímos a identificação e verificação de todos os símbolos reservados da linguagem que ainda não tínhamos tratado.

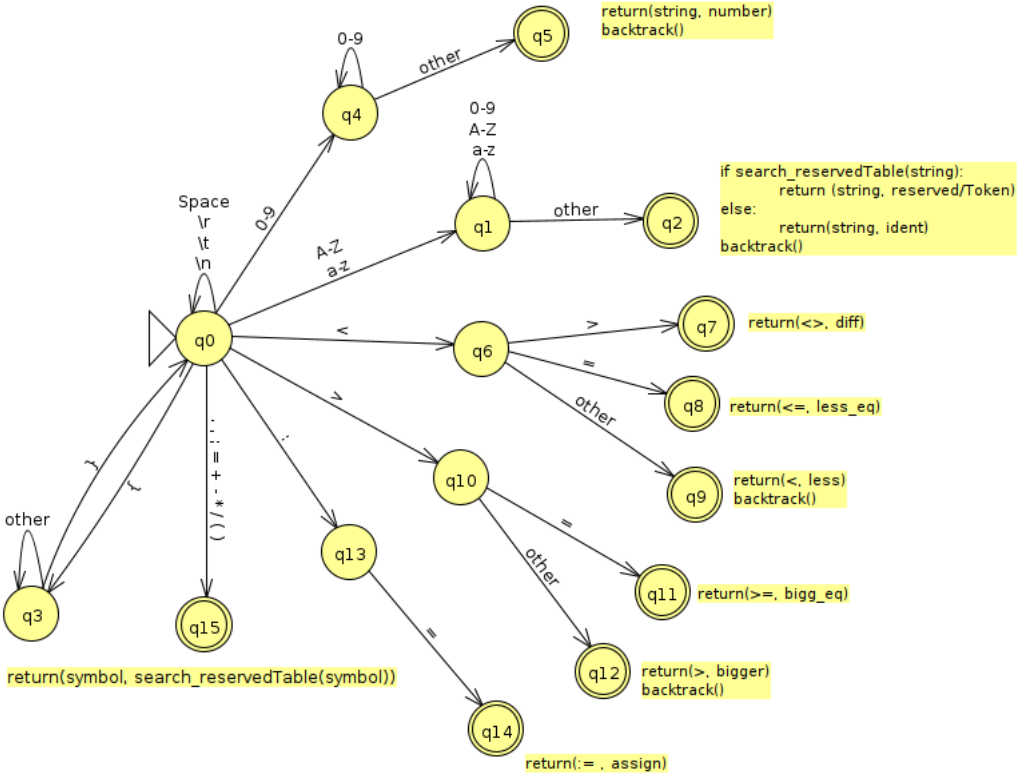

Figura 5: Adição para símbolos reservados.

2.2.6 Autômato final

A figura 2.2.6 representa o autômato final de analise léxica para a linguagem PL/0, nele foram adicionados os 3 estados finais q16, q17, q18, dos quais retornam os seguintes erros: *"Invalid char"* (caracter inválido); *"Malformed assign operation"* (operador de atribuição malformado); e *"Unexpected end of file"* (fim inesperado do arquivo), respectivamente.

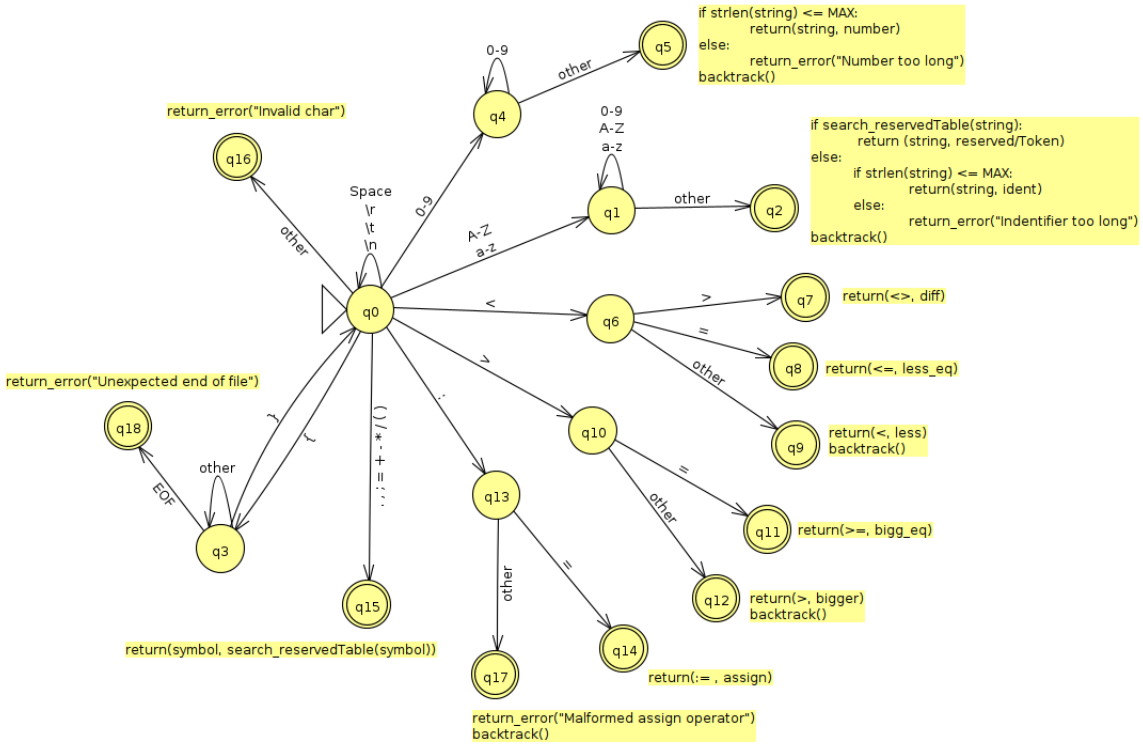

Figura 6: Autômato completo.

3 Instruções para Compilação e Execução

Para compilar e executar o código, feito na linguagem C, do analisador léxico referente ao autômato feito e apresentado na sessão 2 basta seguir os passos das próximas subsessões.

3.1 Compilação

- Em seu terminal de comando, localize a pasta descompactada do projeto
- Primeiramente, deve-se compilar o programa, para isso digite:
  make all

3.2 Execução

- Em seguida, para usar o programa feito e analisar algum arquivo, deve-se executar o programa, o que é feito com o seguinte comando:
  ./pl0-compiler <nome do arquivo para analisar> <arquivo do resultado

da analise>
Por exemplo:
./pl0-compiler pl0_examples/meu_programa.pl0 arq_saida.log
Onde:

- *pl0_examples/* é a pasta onde estão os arquivos escritos na linguagem PL/0 no nosso projeto;
- *meu_programa.pl0* é o arquivo em PL/0 que se deseja analisar, nesse caso é um dos exemplos nesse projeto. Ele não precisa, necessáriamente ter a extensão .pl0, pode ser .txt;
- *arq_saida.log* é o arquivo onde terá a análise léxica, esse arquivo não precisa ser criado anteriormente, o próprio programa irá cria-lo. Ele não precisa ter a extensão .log, pode ser .txt.

3.3 Visualização da análise

- Para se ver o conteúdo do arquivo de saída, onde está a análise, direto pelo terminal, basta digitar o comando:
  cat arq_saida.log

3.4 Flags

Para auxiliar na utilização do programa, existem 3 flags que podem ser aplicadas na execução do programa.

1. **Flag *-v***

   Caso tenha recebido uma mensagem de erro, mas deseja-se que ela seja melhor explicada, basta adicionar a flag *-v* no comando de execução, dessa forma:

   - *./pl0-compiler pl0_examples/meu_programa.pl0 arq_saida.log -v*

   Imagens da utilização dessa flag estão na sessão de exemplos 4.

1. **Flag *-h***

   - A flag *-h* vai exibir uma mensagem de ajuda, onde é explicado quais argumentos tem que ser inseridos ao executar o programa e as flags que podem ser usadas, sua utilização é da forma:
     ./pl0-compiler pl0_examples/meu_programa.pl0 arq_saida.log -h

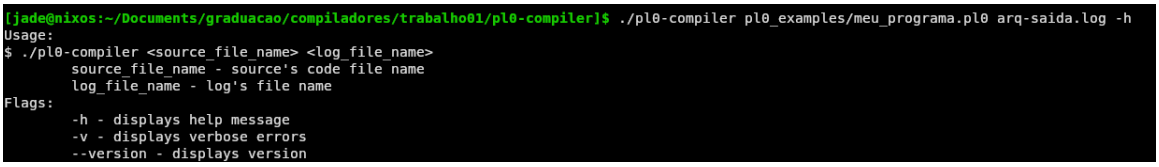

Figura 7: Mensagem exibida no terminal ao usar a flag *-h*.

Essa mesma mensagem será exibida caso se tente executar o programa sem os devidor argumentos.

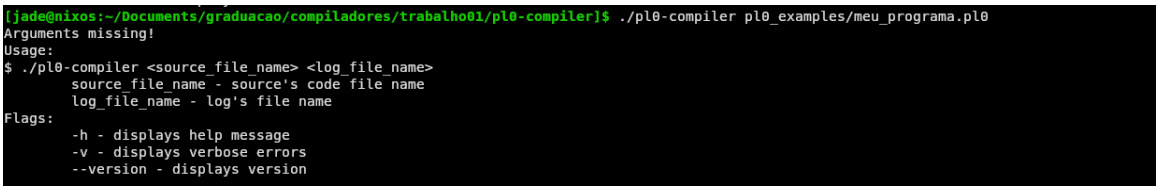

Figura 8: Saída exibida no terminal ao tentar executar o programa sem todos os argumentos necessários.

1. **Flag *–version***

- A flag *–version* mostra a versão atual do programa, ela pode ser usada de forma individual. Apóis compliar o programa, basta digitar o comando: *./pl0-compiler pl0_examples/meu_programa.pl0 arq_saida.log –version* que a versão será exibida na tela.

Figura 9: Mensagem exibida no terminal ao usar a flag *–version*.

Após ver o programa rodar, recomendá-se usar o comando *make clean* para que os arquivos criados na compilação e execução sejam apagados.

4 Exemplos

Aqui vamos apresentar exemplos diferentes para cada um dos 5 tipos de erros que o nosso analisador léxico pode identificar.

4.1 Erro *Found an unexpected end of file!*

Esse erro ocorre quando o sinal que representar fechamento de comentários não ocorre.

Na imagem 4.1 mostra o código onde o sinal de comentários ($\{$) é aberto, mas ele não é fechado, na linha 1.

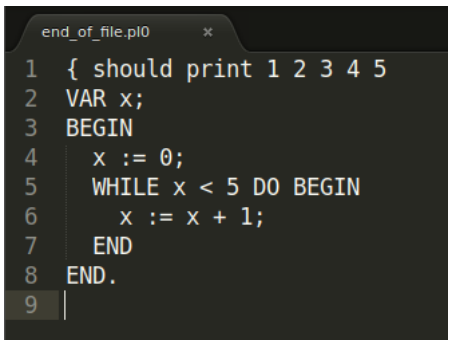

Figura 10: Código que gera o erro "*Found an unexpected end of file*".

Abaixo, temos o conteúdo do arquivo *arq-saida.log*, que é o resultado da análise feita do código 4.1, sem utilização de nenhuma flag.

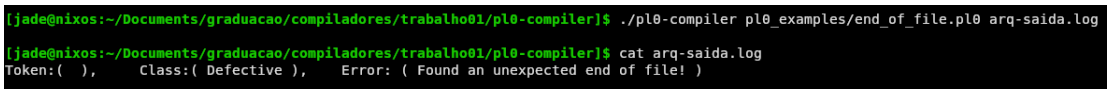

Figura 11: Análise do código com o erro "*Found an unexpected end of file*".

Já na imagem 4.1 temos o conteúdo do arquivo *arq-saida.log* ao usarmos a flag *-v*.

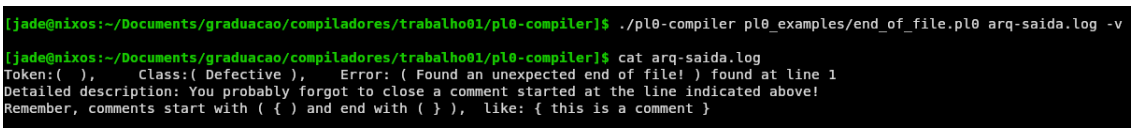

Figura 12: Análise do código com o erro "*Found an unexpected end of file*"usando a flag *-v*.

4.2 Erro *The identifier is too long!*

O erro "*The identifier is too long!*"ocorre quando é digitado um identificador de tamanho maior que o definido no início do nosso programa, nesse caso foi definido uma constante chama *MAX* de tamanho 100.

Na imagem 4.2 temos o código onde o identificador tem seu tamanho maior que 100.

```
identifier_long.pl0
1 BEGIN
2     abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabecdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
3 END.
```

Figura 13: Código que gera o erro de "*The identifier is too long!*".

A seguir temos o conteúdo do aquivo *arq-saida.log* que contem a análise léxica do código acima sem a utilização de nenhuma flag.

```
[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ ./pl0-compiler pl0_examples/identifier_long.pl0 arq-saida.log

[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ cat arq-saida.log
Token:( BEGIN ),        Class:( BEGIN ),
Token:( abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabecdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ ),    Class:( Defective ),    Error: ( The identifier is too long! )
Token:( END ),  Class:( END ),
Token:( . ),    Class:( Dot ),
```

Figura 14: Análise do código com o erro "*The identifier is too long!*".

A imagem 4.2 contem a análise léxica com a utilização da flag *-v*.

```
[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ ./pl0-compiler pl0_examples/identifier_long.pl0 arq-saida.log -v

[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ cat arq-saida.log
Token:( BEGIN ),        Class:( BEGIN ),
Token:( abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabecdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ ),    Class:( Defective ),    Error: ( The identifier is too long! ) found at line 3
Detailed description: The identifier must be made up of a maximum of 100 characters!
Token:( END ),  Class:( END ),
Token:( . ),    Class:( Dot ),
```

Figura 15: Análise do código com o erro "*The identifier is too long!*"usando a flag *-v*.

4.3 Erro *Found a malformed assign operator!*

Esso erro acontece quando é digitado o símbolo *:* e em seguida é digitado qualquer outro símbolo que não é *=*.

A imagem 4.3 é o código onde o erro ocorre.

```
malformed_operator.pl0
1 BEGIN :
2     1 + 2 * 3
3 END.
4
```

Figura 16: Código que gera o erro "*Found a malformed assign operator!*".

Abaixo, é exibido a análise léxica que está no arquivo *arq-saida.log*.

```
[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ ./pl0-compiler pl0_examples/malformed_operator.pl0 arq-saida.log

[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ cat arq-saida.log
Token:( BEGIN ),        Class:( BEGIN ),
Token:( : ),    Class:( Defective ),    Error: ( Found a malformed assign operator! )
Token:( 1 ),    Class:( Number ),
Token:( + ),    Class:( Plus ),
Token:( 2 ),    Class:( Number ),
Token:( * ),    Class:( Multiplication ),
Token:( 3 ),    Class:( Number ),
Token:( END ),  Class:( END ),
Token:( . ),    Class:( Dot ),
```

Figura 17: Análise do código com o erro "*Found a malformed assign operator!*".

Na imagem 4.3 é a análise léxica feita pelo programa utilizando a flag *-v*.

```
[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ ./pl0-compiler pl0_examples/malformed_operator.pl0 arq-saida.log -v

[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ cat arq-saida.log
Token:( BEGIN ),        Class:( BEGIN ),
Token:( : ),    Class:( Defective ),    Error: ( Found a malformed assign operator! ) found at line 2
Detailed description: An assign operator should be like ( := ) !
Token:( 1 ),    Class:( Number ),
Token:( + ),    Class:( Plus ),
Token:( 2 ),    Class:( Number ),
Token:( * ),    Class:( Multiplication ),
Token:( 3 ),    Class:( Number ),
Token:( END ),  Class:( END ),
Token:( . ),    Class:( Dot ),
```

Figura 18: Análise do código com o erro "*Found a malformed assign operator!*"utilizando a flag *-v*.

4.4 Erro *The character is invalid!*

Esse erro ocorre quando é inserido qualquer caracter que não consta no autômato final feito (2.2.6).

Abaixo tem-se o código que causa o erro "*The character is invalid*".

meu_programa.pl0 ×

```
1  VAR a,b,c;
2  BEGIN
3      a:=2;
4      b:=3;
5      c:=@+b
6  END.
```

Figura 19: Código que gera o erro "*The character is invalid*".

Nas imagens 4.4 e 4.4 são as análises léxicas feita do código acima, sem a utilização da flag *-v* e com a flag, respectivamente.

```
[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ ./pl0-compiler pl0_examples/meu_programa.pl0 arq-saida.log

[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ cat arq-saida.log
Token:( VAR ),  Class:( VAR ),
Token:( a ),    Class:( Identifier ),
Token:( , ),    Class:( Comma ),
Token:( b ),    Class:( Identifier ),
Token:( , ),    Class:( Comma ),
Token:( c ),    Class:( Identifier ),
Token:( ; ),    Class:( Semicolon ),
Token:( BEGIN ),        Class:( BEGIN ),
Token:( a ),    Class:( Identifier ),
Token:( := ),   Class:( Assign ),
Token:( 2 ),    Class:( Number ),
Token:( ; ),    Class:( Semicolon ),
Token:( b ),    Class:( Identifier ),
Token:( := ),   Class:( Assign ),
Token:( 3 ),    Class:( Number ),
Token:( ; ),    Class:( Semicolon ),
Token:( c ),    Class:( Identifier ),
Token:( := ),   Class:( Assign ),
Token:( @ ),    Class:( Defective ),    Error: ( The character is invalid! )
Token:( + ),    Class:( Plus ),
Token:( b ),    Class:( Identifier ),
Token:( END ),  Class:( END ),
Token:( . ),    Class:( Dot ),
```

Figura 20: Análise do código com o erro "*The character is invalid*".

```
[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ ./pl0-compiler pl0_examples/meu_programa.pl0 arq-saida.log -v

[jade@nixos:~/Documents/graduacao/compiladores/trabalho01/pl0-compiler]$ cat arq-saida.log
Token:( VAR ),  Class:( VAR ),
Token:( a ),    Class:( Identifier ),
Token:( , ),    Class:( Comma ),
Token:( b ),    Class:( Identifier ),
Token:( , ),    Class:( Comma ),
Token:( c ),    Class:( Identifier ),
Token:( ; ),    Class:( Semicolon ),
Token:( BEGIN ),        Class:( BEGIN ),
Token:( a ),    Class:( Identifier ),
Token:( := ),   Class:( Assign ),
Token:( 2 ),    Class:( Number ),
Token:( ; ),    Class:( Semicolon ),
Token:( b ),    Class:( Identifier ),
Token:( := ),   Class:( Assign ),
Token:( 3 ),    Class:( Number ),
Token:( ; ),    Class:( Semicolon ),
Token:( c ),    Class:( Identifier ),
Token:( := ),   Class:( Assign ),
Token:( @ ),    Class:( Defective ),    Error: ( The character is invalid! ) found at line 5
Detailed description: The accepted characters are:
        - letters (capitalize or not) from a to z ([a-z] and [A-Z])
        - digits from 0 to 9
        - math symbols ( +, -, >, =, <, /, * )
        - ponctuation symbols like: the comma [ , ] and [ ), (, ., ;, :, }, { ]
        - invisible characters ('\n', '\r', '\t', ' ')
Token:( + ),    Class:( Plus ),
Token:( b ),    Class:( Identifier ),
Token:( END ),  Class:( END ),
Token:( . ),    Class:( Dot ),
```

Figura 21: Análise do código com o erro "*The character is invalid*"utilizando a flag *-v*.

4.5 Erro *The number is too long!*

O código da imagem abaixo, gera o erro "*The number is too long!*", que ocorre quando o número tem tamanho maior que da constante chamada *MAX* de tamanho 100, definida no início do programa.

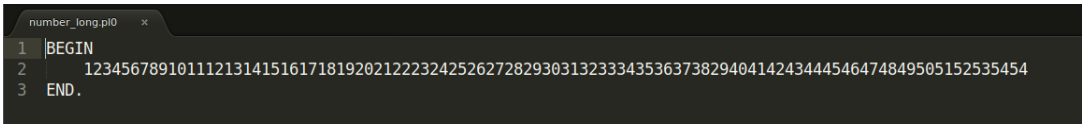

Figura 22: Código que vai gerar o erro "*The number is too long!*".

Nas imagens 4.5 e 4.5 são, respectivamente, o conteúdo do arquivo *arq-saida.log* com a análise léxica sem a utilização da flag *-v* e com a utilização dessa flag.

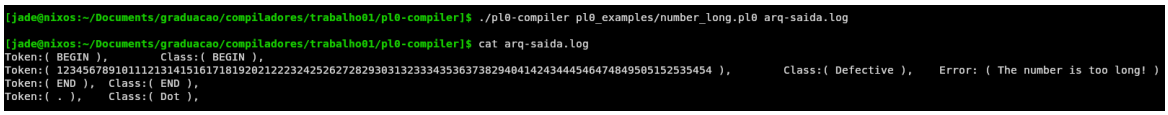

Figura 23: Análise do código com o erro "*The number is too long!*".

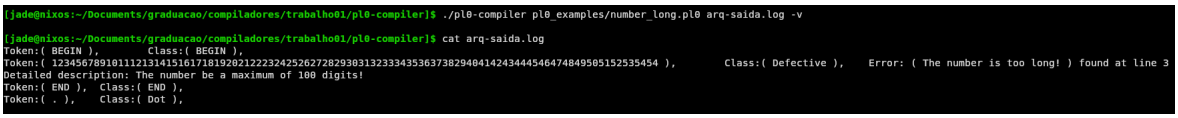

Figura 24: Análise do código com o erro "*The number is too long!*"utilizando a flag *-v*.

Referências

[1] Thiago Pardo, Prof. *Aula 2 - visão geral de um compilador.*

[2] Thiago Pardo, Prof. *Aula 3 - compiladores, linguagens formais e seus mecanismos.*

[3] Thiago Pardo, Prof. *Aula 4 - introdução às linguagens formais.*

[4] Thiago Pardo, Prof. *Aula 5 - autômatos finitos determinísticos - parte 1.*

[5] Thiago Pardo, Prof. *Aula 6 - autômatos finitos determinísticos - parte 2.*

[6] Thiago Pardo, Prof. *Aula 7 - autômatos finitos não determinísticos.*

[7] Thiago Pardo, Prof. *Aula 8 - gramáticas regulares.*

[8] Thiago Pardo, Prof. *Aula 9 - gramáticas regulares.*

[9] Thiago Pardo, Prof. *Aula 10a - análise léxica.*

[10] Thiago Pardo, Prof. *Aula 10b - expressões regulares.*

[11] Thiago Pardo, Prof. *Aula 11 - prática com PL/0 e expressões regulares.*

[12] Thiago Pardo, Prof. *Especificação do trabalho 1: análise léxica*

[13] Orion Transfer *PL0 Language Tools - Tests* `https://github.com/oriontransfer/PL0-Language-Tools/tree/master/tests`