PCS2056 – Linguagens e Compiladores

Rogério Yuuki Motisuki - 8587052

Guilherme Mamprin - 8587584

1. Quais são as funções do analisador léxico nos compiladores/interpretadores?

O analisador léxico é responsável por receber uma entrada textual e reconhecer símbolos da linguagem, de forma a preparar para a fase da **análise sintática**.

O resultado da análise léxica é uma lista de símbolos (tokens, átomos) e suas propriedades (posição, classe, etc).

2. Quais as vantagens e desvantagens da implementação do analisador léxico como uma fase separada do processamento da linguagem de programação em relação à sua implementação como sub-rotina que vai extraindo um átomo a cada chamada?

Arquiteturalmente, a separação da análise léxica em uma fase é vantajoso por abstrair uma responsabilidade do programa principal. É o chamado *"Separation of concerns"*.

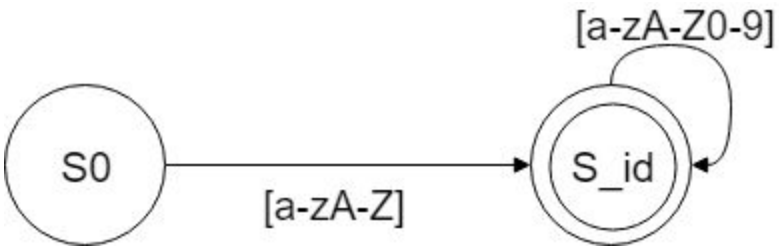
A desvantagem é o aumento de complexidade do projeto e a comunicação entre módulos.

3. Defina formalmente, através de expressões regulares sobre o conjunto de caracteres ASCII, a sintaxe de cada um dos tipos de átomos a serem extraídos do texto-fonte pelo analisador léxico, bem como de cada um dos espaçadores e comentários.

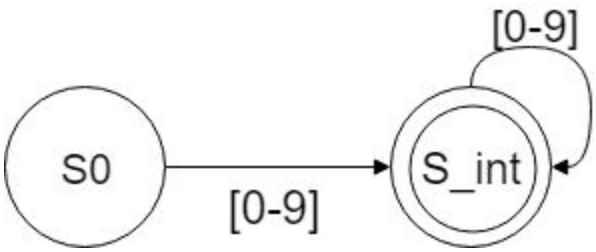
Token	**Expressão Regular**
Identificador	`[a-zA-Z][a-zA-Z0-9]*`
Número Inteiro	`[0-9]+`
Número Decimal	`[0-9]+\.[0-9]*|\.[0-9]+`
String	`".*"|'.*'`
Palavras reservadas	`if|else|case|default|break|for|while|continue|function|return|switch|int|float|string|void|struct|typedef`
Símbolos Especiais	`={1,2}|!=?|>=?|<=?|\+|-|\*|\/|&|,|;|||{|}|\(|\)`
Comentário	`\/\/.*|\/\*(.|[\n\r])*\*\/`
Espaçadores	`[ \t\n\r]*`

4. Converta cada uma das expressões regulares, assim obtidas, em autômatos finitos equivalentes que reconheçam as correspondentes linguagens por elas definidas.

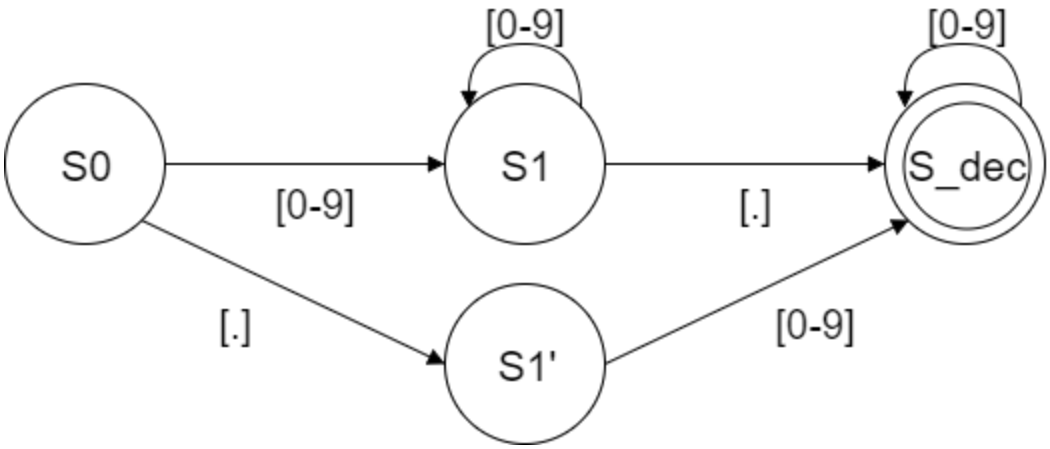
Identificador:



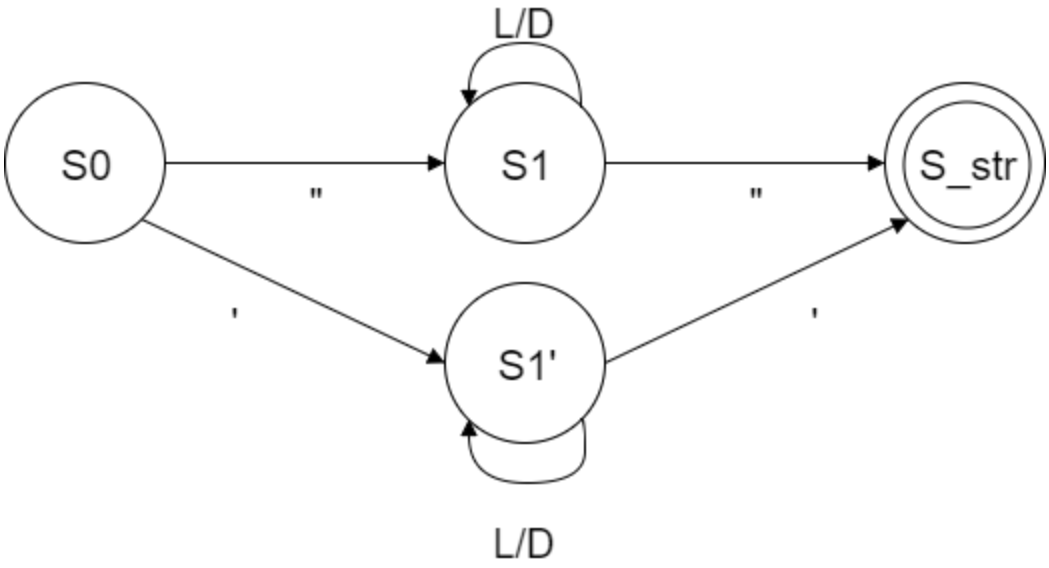
Número Inteiro:



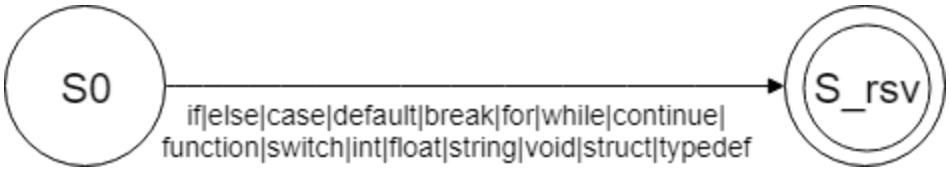
Número Decimal:



String

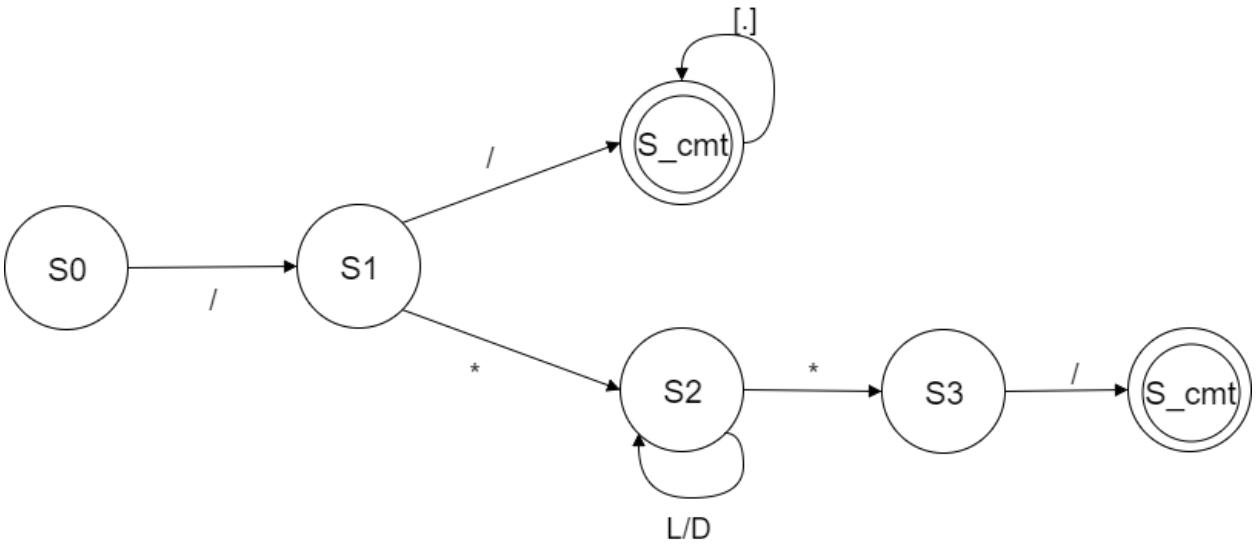


Palavras Reservadas

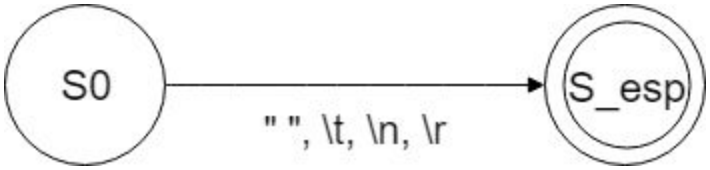


Símbolos Especiais

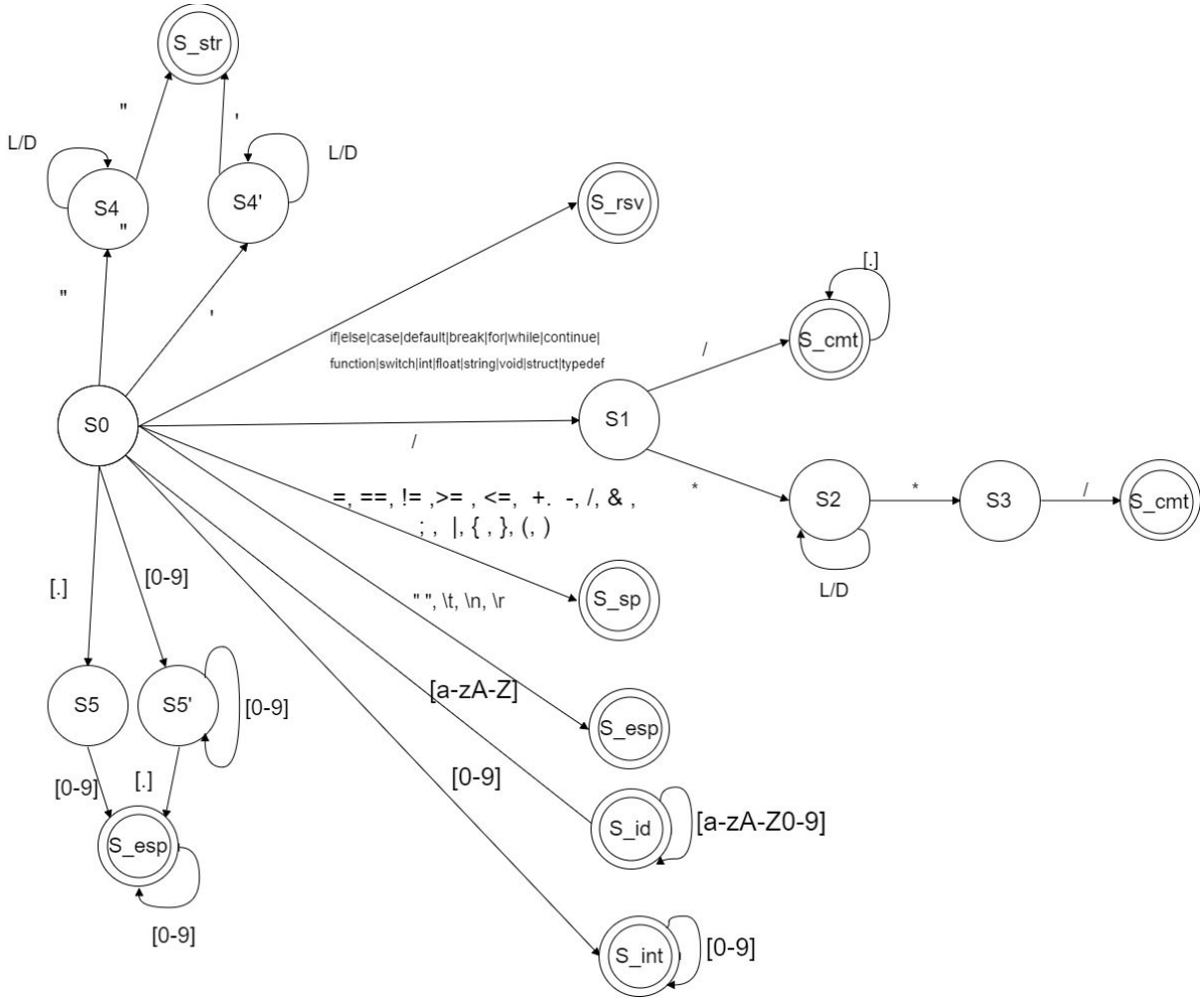
Comentário



Espaçadores

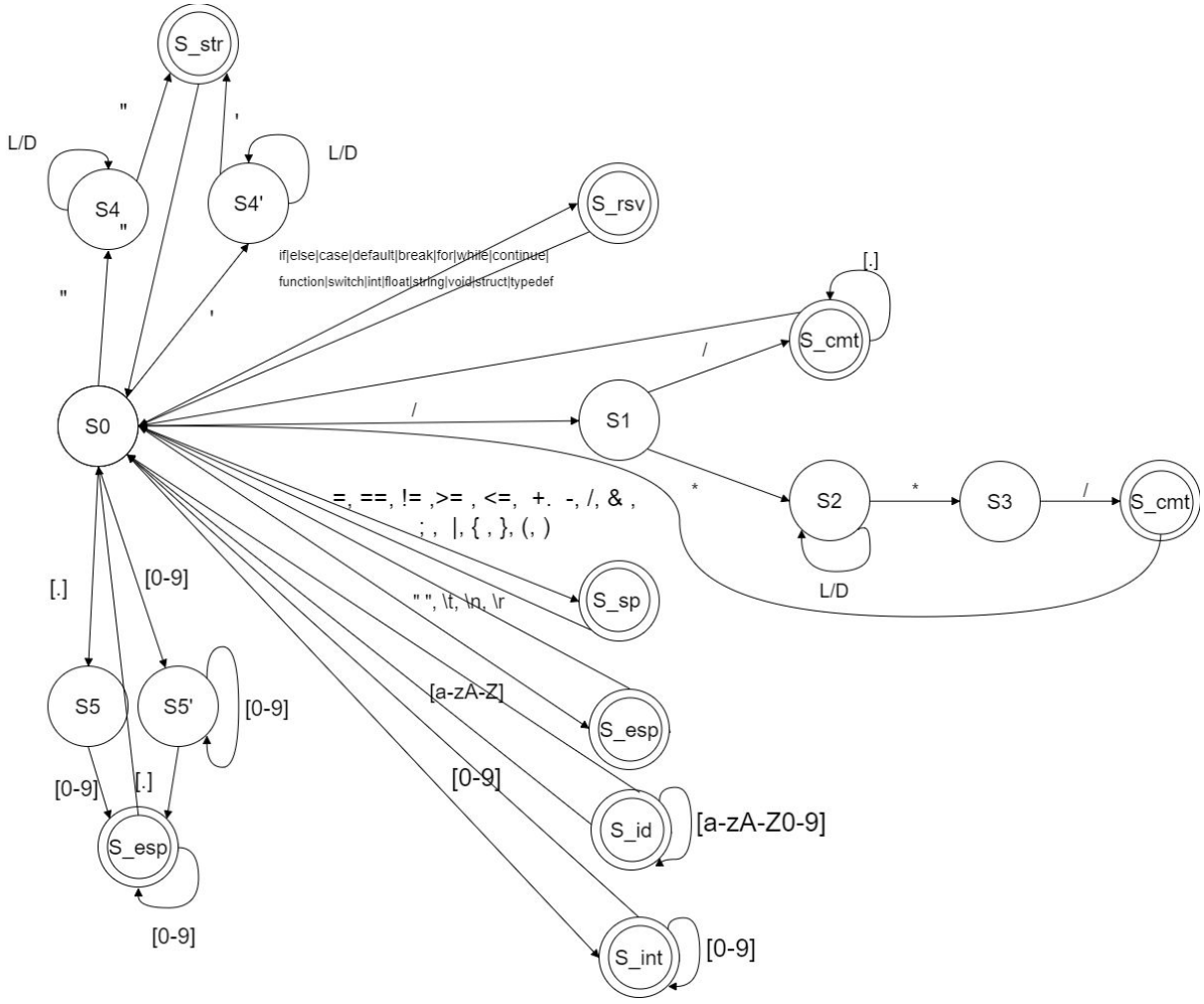


5. Crie um autômato único que aceite todas essas linguagens a partir de um mesmo estado inicial, mas que apresente um estado final diferenciado para cada uma delas.



6. Transforme o autômato assim obtido em um transdutor, que emita como saída o átomo encontrado ao abandonar cada um dos estados finais para iniciar o reconhecimento de mais um átomo do texto.

Modificando o autômato anterior para que a partir dos estados finais ele retorne aos estados iniciais ele se torna um transdutor, para evitar poluir demais o desenho, considera-se que ao retornar ao estado inicial a partir de um dos estados finais, o autômato emite o átomo encontrado.



7. Converta o transdutor assim obtido em uma sub-rotina, escrita na linguagem de programação de sua preferência. Não se esqueça que o final de cada átomo é determinado ao ser encontrado o primeiro símbolo do átomo ou do espaçador seguinte. Esse símbolo não pode ser perdido, devendo-se, portanto, tomar os cuidados de programação que forem necessários para reprocessá-los, apesar de já terem sido lidos pelo autômato.

Foi utilizado o Flex para gerar o analisador léxico. O arquivo de entrada .flex utilizado foi o seguinte:

```
%option noyywrap

%{
#include "main.h"
#define YY_DECL Token yylex()
%}

LETRA [a-zA-Z]
NUMERO [0-9]
LETRA_NUMERO {LETRA}|{NUMERO}

COMENTARIO \/\/.*|\/\*(.|[\n\r])*\*\/
ESPACADOR [ \t\n\r]

IDENTIFIER {LETRA}{LETRA_NUMERO}*
INTEGER {NUMERO}+
DECIMAL {NUMERO}+\.{NUMERO}*|\.{NUMERO}+
STRING ".*"|'.*'
KEYWORD
if|else|case|default|break|for|while|continue|function|return|switch|in
t|float|string|void|struct|typedef
SYMBOL =\{1,2\}|!=?|>=?|<=?|\+|-|\*|\/|&|,|;|\||\{|\}|\(|\)

%%

{COMENTARIO} ;
{ESPACADOR} ;

{KEYWORD} { return CREATE_TOKEN(KEYWORD, yytext); }
{IDENTIFIER} { return CREATE_TOKEN(IDENTIFIER, yytext); }
{INTEGER} { return CREATE_TOKEN(INTEGER, yytext); }
{DECIMAL} { return CREATE_TOKEN(DECIMAL, yytext); }
{STRING} { return CREATE_TOKEN(STRING, yytext); }
{SYMBOL} { return CREATE_TOKEN(SYMBOL, yytext); }

<<EOF>> { return CREATE_TOKEN(EOF_TOKEN, yytext); }

%%
```

8. Crie um programa principal que chame repetidamente a sub-rotina assim construída, e a aplique sobre um arquivo do tipo texto contendo o texto-fonte a ser analisado. Após cada chamada, esse programa principal deve imprimir as duas componentes do átomo extraído (o tipo e o valor do átomo encontrado). Faça o programa parar quando o programa principal receber do analisador léxico um átomo especial indicativo da ausência de novos átomos no texto de entrada.

O programa principal encontra-se nos arquivos main.c e main.h em anexo.

9. Relate detalhadamente o funcionamento do analisador léxico assim construído, incluindo no relatório: descrição teórica do programa; descrição da sua estrutura; descrição de seu funcionamento; descrição dos testes realizados e das saídas obtidas.

Nas ações do reconhecimento léxico via Flex, foi executado uma função de criador de structs Token. A struct Token possui um enum de tipo, e uma string de valor.

A entrada de teste utilizada foi:

```
1. teste
2.
3. 12
4.
5. 772. 61.321
6.
7. .90
8.
9. // comentario
10.
11.
12. /*
13. comentario
14. */
15.
16. function teste(int *ok) {
17.     return !ok;
18. }
```

A saída obtida foi:

```
>  Token IDENTIFIER - teste
>  Token INTEGER - 12
>  Token DECIMAL - 772.
>  Token DECIMAL - 61.321
>  Token DECIMAL - .90
```

```
>  Token KEYWORD - function
>  Token IDENTIFIER - teste
>  Token SYMBOL - (
>  Token KEYWORD - int
>  Token SYMBOL - *
>  Token IDENTIFIER - ok
>  Token SYMBOL - )
>  Token SYMBOL - {
>  Token KEYWORD - return
>  Token SYMBOL - !
>  Token IDENTIFIER - ok
>  Token SYMBOL - ;
>  Token SYMBOL - }
```

10.Explique como enriquecer esse analisador léxico com um expansor de macros do tipo #DEFINE, não paramétrico nem recursivo, mas que permita a qualquer macro chamar outras macros, de forma não cíclica. (O expansor de macros não precisa ser implementado).

Uma maneira seria construir uma tabela de macros, mapeando um token a uma lista de tokens, e substituir tokens reconhecidos caso uma entrada de macro correspondente seja encontrada.

Para as macros chamarem outras macros, o processo de substituição de token deve ocorrer também no momento de construção da tabela, utilizando o estado da tabela atual.

Dessa forma, macros podem chamar apenas macros previamente definidos, impedindo ciclos e recursão.