**Questões**

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

**O analisador léxico possui várias funções no projeto de um compilador. A principal delas é a extração dos elementos básicos da linguagem em cadeias de caracteres mapeadas para outras de tamanho fixo. Esse mapeamento gera os chamados átomos. Dentre os elementos básicos da lingugem, existem os identificadores e as palavras reservadas.**

**Além disso, ele é responsável também pela remoção de delimitadores e comentários do código; pela conversão numérica, padronizando os diversos valores numéricos de tipos diferentes encontrados no código para um tipo específico; pela identificação de erros de cadeias de caracteres que não seguem as regras de formação dos átomos.**

**Outras funções auxiliares podem estar presentes, como listagens e gerações de tabelas e referências cruzadas, entre outras.**

**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?**

**Uma das diferenças entre a abordagem do analisador léxico como uma fase separada e como sub-rotina é que enquanto na primeira deve-se guardar o resultado da análise em um arquivo que será usado posteriormente pelo analisador sintático, a segunda gera átomos conforme a necessidade dele e passa-os como parâmetro. Neste ponto, a performance da segunda abordagem é melhor, além de a compilação se dar em um passo único.**

**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.**

* ****Palavras reservadas:****

**PR = int|caract|pflut|bool|comp|deftipo|func|se|senao|enquanto|para**

* ****Números:****

**DIG = [0-9]**

**NFLOAT = (DIG)\*’.’(DIG)+**

**NUM = DIG+|NFLOAT**

* ****Identificadores:****

**LETRA = [A-Za-z]**

**ID = LETRA(LETRA|DIG)\***

* ****Pontuação, operação e separação:****

**SINAIS = ;|{|}|(|)|[|]|!|+|-|\*|/|%|=|<|>|&**

* ****Sinais compostos:****

**SINAISC = ’:=’ | ’||’**

* ****Separadores:****

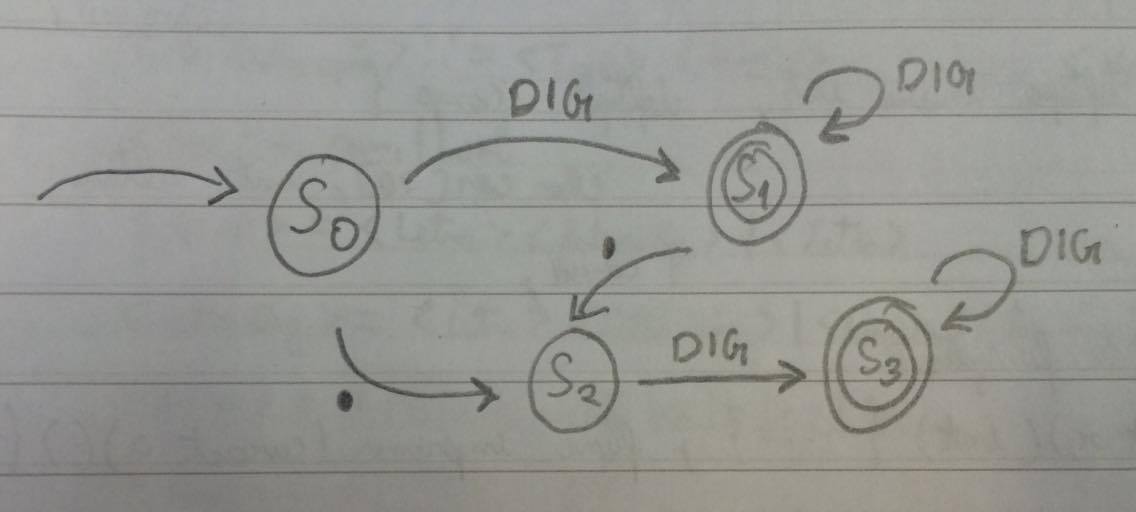
**COMENT = ’/\*’(.\*^’\*/’)’\*/’ -> /\* (qualquer coisa menos \*/) \*/**

**COMENTL = ’//’(.\*^’\n’)’\n’ -> // (qualquer coisa menos quebra de linha) quebra de linha**

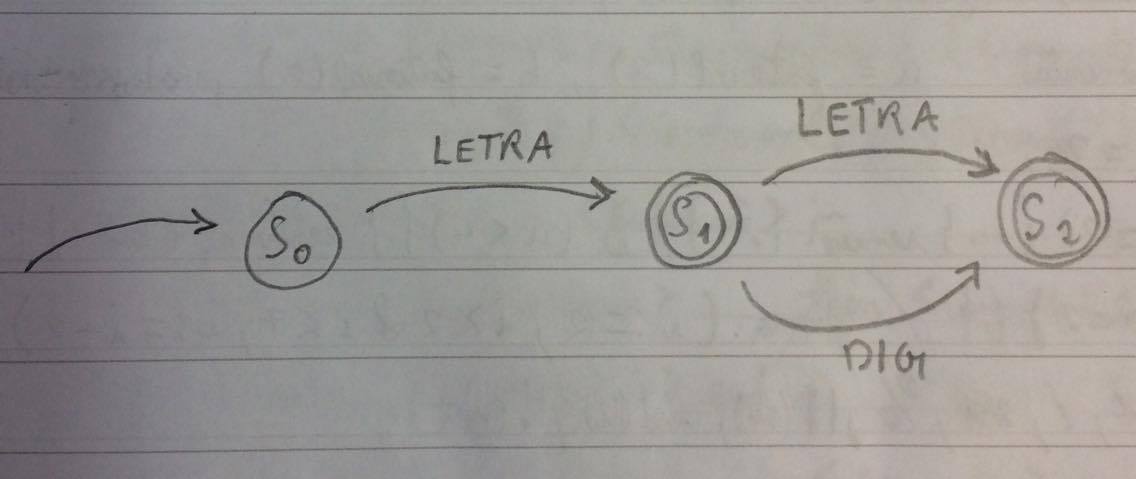
**SEPARADORES = ’\n’|’ ’ (quebra de linha e espaço)**

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

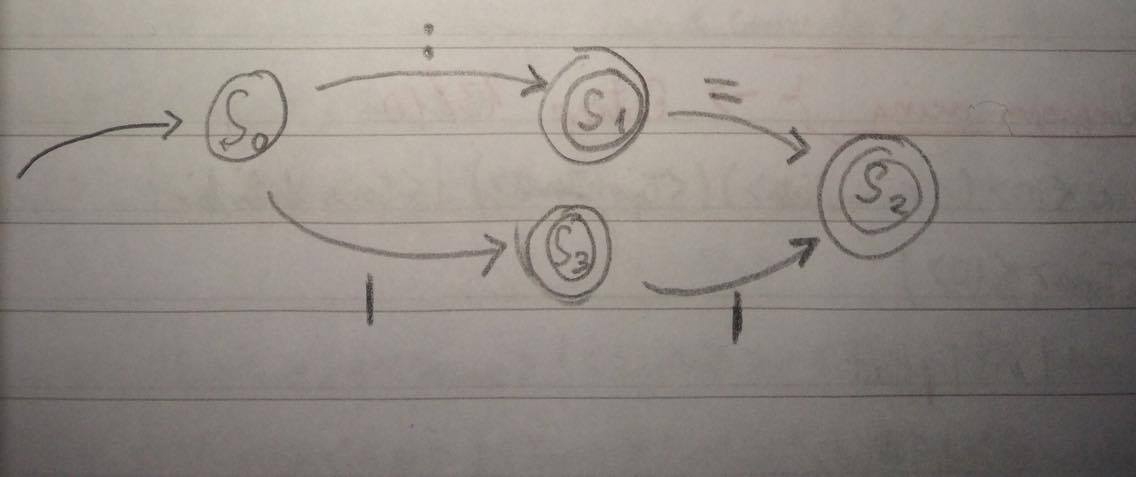
**Números:**



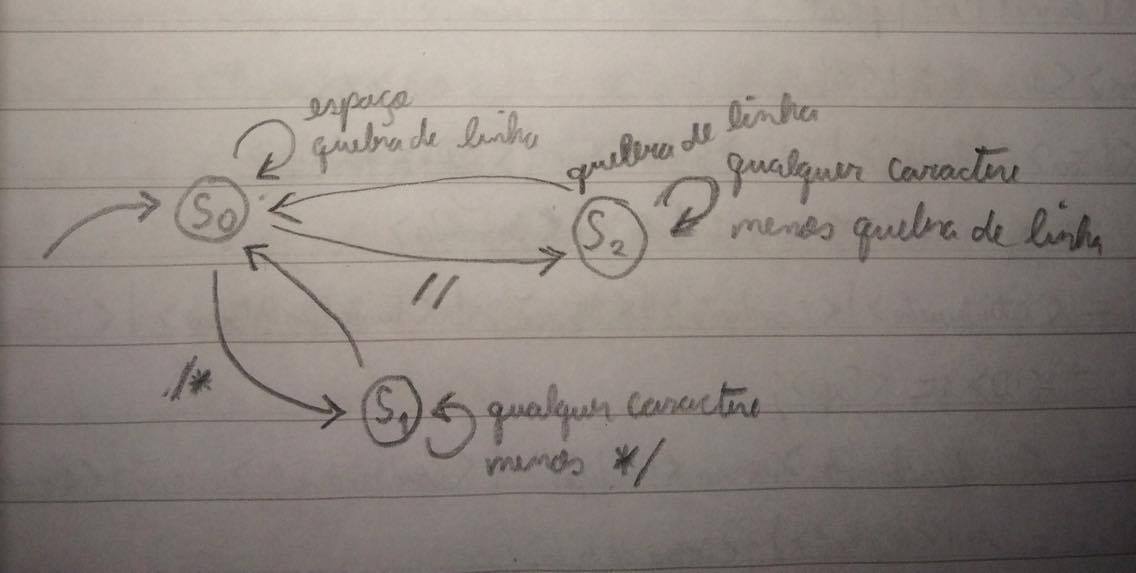
**Identificadores:**



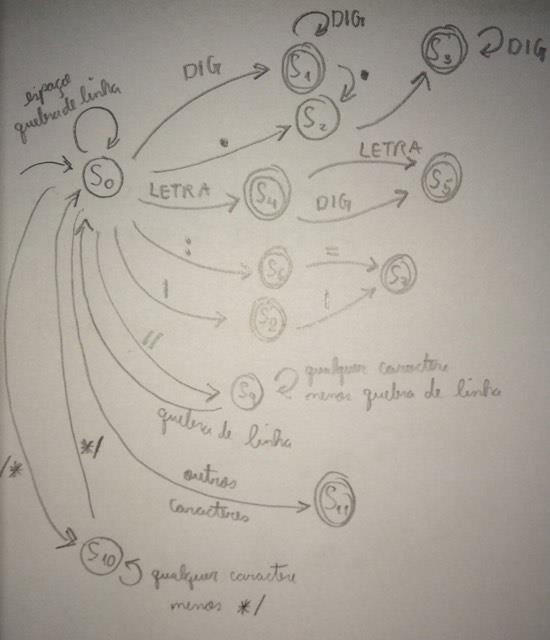
**Sinais compostos:**



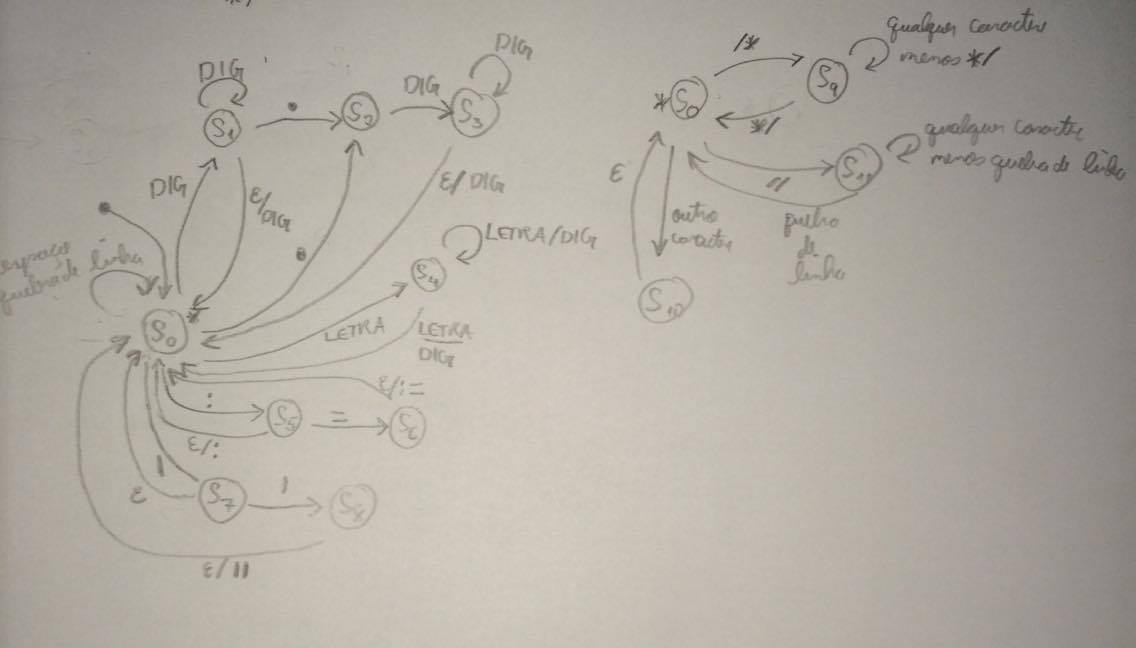
**Separadores:**



**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.**



**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.**

**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.**

**Questões 7 e 8 entregues no projeto.**

**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.**

**O programa é uma função principal que chama repetidas vezes uma subrotina que vai lendo o arquivo de entrada, guarda o próximo átomo a ser mandado por meio de verificações e depois retorna o tipo dele por meio de uma tabela. A linguagem de entrada é uma linguagem com sintaxe similar ao C, mas com algumas modificações.**

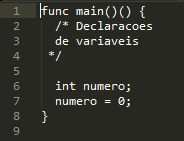
**O programa principal, localizado no arquivo “Lex.c” chama a sub-rotina “getToken” repetidamente. Ela recebe o arquivo a ser lido, um ponteiro de uma string para atribuir o valor do token e retorna o tipo dele, com o seguinte mapeamento:**

**0 – Identificador, 1 – Palavra reservada, 2 – Número, 3 – Sinal, 4 - Outro**

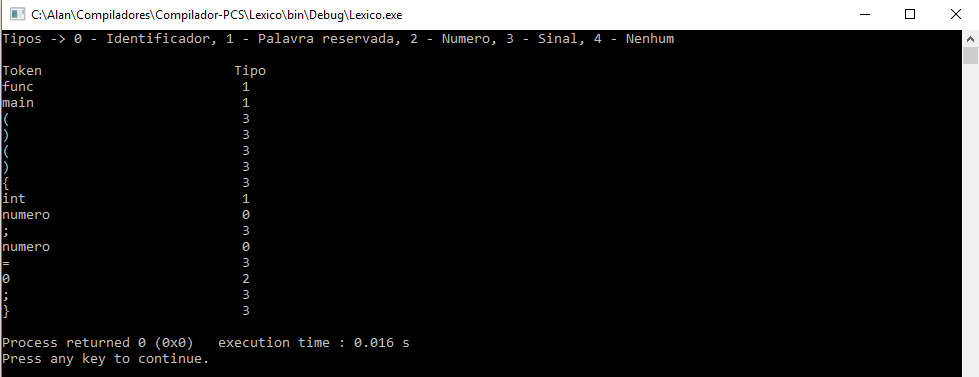
**A subrotina lê caractere a caractere do arquivo e realiza verificações muito similares ao trandutor da questão 6, ignorando comentários, brancos e quebras de linhas e verificando qual o átomo a ser guardado na string que recebeu. Depois que o átomo é montado, ele chama a subrotina “getTokenType” do arquivo “Table.c”, que contém uma tabela com as palavras reservadas e os sinais. Se necessário, ele percorre a tabela e devolve o tipo do átomo que foi passado.**

**Depois do retorno de cada chamada da função “getToken”, o programa principal imprime o átomo encontrado e seu tipo. Três testes principais foram realizados, utilizando os seguintes arquivos de entrada:**

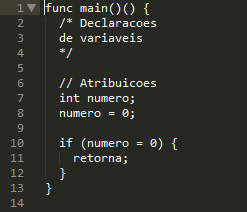
* **Exemplo1.txt:**



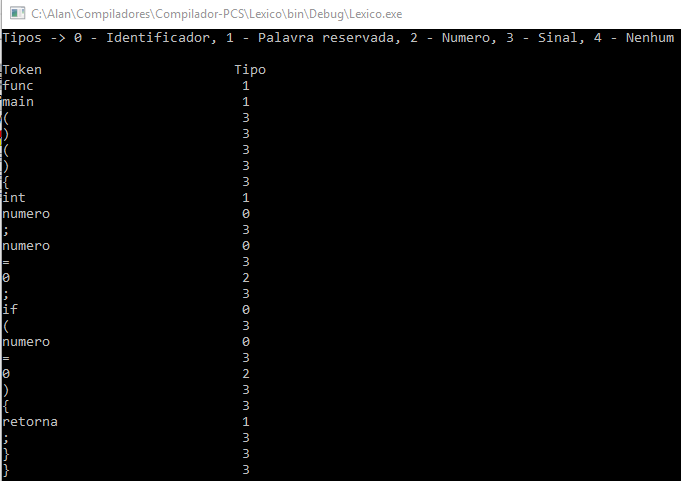
**Resultado:**



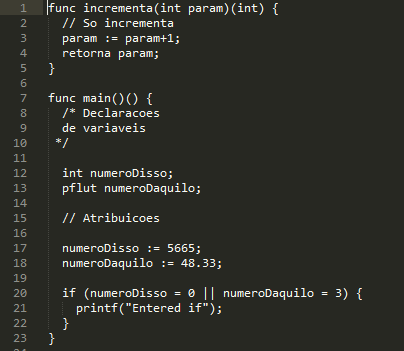
* **Exemplo2.txt:**



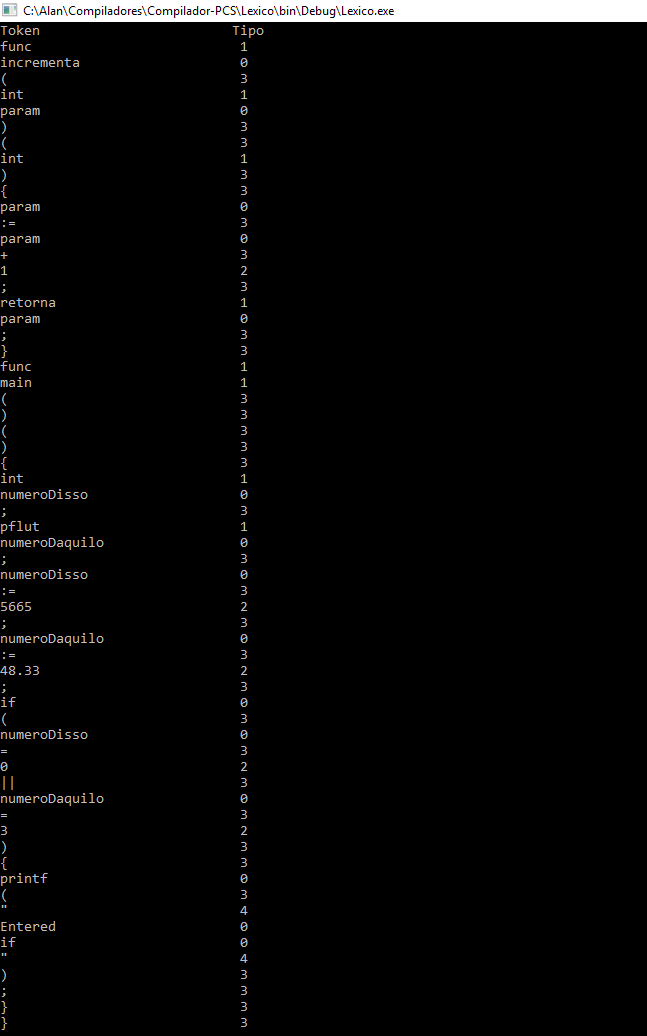
**Resultado:**



* **ENTRADA.txt:**



**Resultado**



**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 possível implementação seria utilizar palavras reservadas para as macros e uma outra palavra reservada, que poderia ser, por exemplo, um caractere que ainda não tem utilização definida na linguagem, para indicar o final das macros. Assim, reliza-se normalmente a extração dos átomos, e o analisador sintático sabe exatamente quando uma macro é iniciada e quando ela termina.**