ANALISE LÉXICA E TABELA DE SÍMBOLOS

Descrição do trabalho:

Nesta etapa, você deverá implementar um analisador léxico para a linguagem MicroPascal (μ-Pascal) cuja descrição fora mostrada em sala e disponibilizada para download.

Seu analisador léxico deverá ser implementado conforme visto em sala de aula, com o auxílio de um autômato finito determinístico. Ele deverá reconhecer um lexema e retornar, a cada chamada, uma struct token, representando o token reconhecido de acordo com o lexema encontrado.

Para facilitar a implementação, uma Tabela de Símbolos (TS) deverá ser usada. Essa tabela conterá, inicialmente, todas as palavras reservadas da linguagem. À medida que novos tokens forem sendo reconhecidos, esses deverão ser consultados na TS antes de serem cadastrados e retornados. Ademais, somente palavras reservadas e identificadores serão cadastrados na TS, não sendo permitido o cadastro de um mesmo token mais de uma vez na TS.

Resumindo, seu Analisador Léxico deverá gerar um arquivo (com a extensão **.lex**) com a lista de todos os tokens reconhecidos, assim como mostrar o que está cadastrado na Tabela de Símbolos. No arquivo, deverá aparecer a tupla <nome, lexema> assim como linha e coluna do token.

Além de reconhecer os tokens da linguagem, seu analisador léxico devera detectar possíveis erros e reportá-los ao usuário. O programa deverá informar o erro e o local onde ocorreu (linha e coluna), lembrando que em análise léxica tem-se 3 tipos de erros: caracteres desconhecidos (não esperados ou inválidos), string nao-fechada antes de quebra de linha e comentário não-fechado antes do fim de arquivo (este não cabível ao Micro Pascal).

Espaços em branco, tabulações, quebras de linhas e comentários não são tokens, ou seja, devem ser descartados/ignorados pelo referido analisador.

O que entregar?

Você deverá entregar nesta etapa:

1. Uma figura apresentando o Autômato Finito Determinístico para reconhecimento dos tokens, conforme visto em sala de aula (de uma olhada na ferramenta JFLAP: http://www.jflap.org/);

2. Todos os arquivos fonte;

3. Relatório técnico contendo explicações do proposito de todas as structs e funções, assim como testes realizados com programas corretos e errados (no mínimo, 3 certos e 3 errados). Os programas testes deverão ser definidos de acordo com a gramática do MicroPascal.

Os resultados deverão apresentar a saída do Analisador Léxico (a sequência de tokens identificados e o local de sua ocorrência) e os símbolos instalados na Tabela de Símbolos, bem como os erros léxicos encontrados.

Regras:

- O trabalho poderá deverá ser desenvolvido em grupos de até 5 alunos;
- Não é permitido o uso de ferramentas para geração do analisador léxico;
- A implementação deverá ser realizada preferencialmente em C;
- Trabalhos total ou parcialmente iguais receberão avaliação nula;
- Ultrapassados cinco (5) dias da data definida para entrega, nenhum trabalho será recebido.

Nomes para os tokens:

Sugestão para nomes de tokens:

Operadores:

- OP_EQ: =
- OP_GE: >=
- OP_MUL: *
- OP_NE: <>
- OP_LE: <=
- OP_DIV: /
- OP_GT: >
- OP_AD: +
- OP_ASS: =
- OP_LT: <
- OP_MIN: -

Simbolos:

- SMB_OBC: {
- SMB_COM: ,
- SMB_CBC: }
- SMB_SEM: ;
- SMB_OPA: (
- SMB_CPA:)

Palavras-chave: program, var, integer, real, begin, end, if, then, else, while, do

Identificadores: ID

Outras características de MicroPascal

- As palavras-chave de MicroPascal são reservadas;
- Toda variável deve ser declarada antes do seu uso;
- A linguagem não permite o uso de comentários;
- A semântica dos demais comandos e expressões e a tradicional do Pascal;
- A linguagem não é case-sensitive;

Exemplo de programa:

O programa a seguir se chama Exemplo, as variáveis x e y são declaradas como do tipo integer, enquanto z é do tipo real. Além disso, o bloco principal do programa contém:

- Atribuições às variáveis (x, y, z).
- Um comando condicional if que compara x e y.
- Um laço while que multiplica z por 1.5 até que z seja maior que 100, ao mesmo tempo que incrementa x.

```
program Exemplo;

var
   x, y : integer;
   z : real;

begin
   x := 10;
   y := 20;
   z := x + y * 2.5;

   if x > y then
      x := x - 1
   else
      y := y + 1;

   while z <= 100 do
   begin
      z := z * 1.5;
      x := x + 2
   end
end.
```