Universidade Estadual do Sudoeste da Bahia – UESB
Departamento de Ciências Exatas – DCET
Disciplina: *Compiladores* **Professores:** *José Carlos Martins Oliveira*
Curso: *Ciência da Computação* **Ano:** *2020* **Período:** *II*
jcarlos@uesb.edu.br

Trabalho prático da I unidade
Implementação do Analisador Léxico - Valor 4,0

1. Especificação do Trabalho
Este trabalho tem por finalidade construir um analisador léxico para uma linguagem de programação que corresponde à uma versão simplificada da linguagem **Pascal (Mini_Pascal)**. Inicialmente, deve-se ter em mente quais estruturas e dados o compilador deve aceitar.

Características da linguagem
1. Suporta os tipos: integer, real, char, string
2. Comandos:
   a. Atribuição com o operador ":="
   b. Entrada com o comando **read**( )
   c. Saída com os comandos **write**( ) e **writeln**( )
   d. Condicional com o comando **if** - **then** - **else**
   e. Repetições com os comandos: **while - do**, **repeat - until**, **for -to - do**
3. Constantes caracteres delimitados por (' ') e constantes strings por (" ")
4. Operadores relacionais: "=", ">=", "<=", ">", "<", "<>"
5. Operadores lógicos: and, or, not
6. Operadores aritméticos: "+", "-", "*", "/" e mod
7. Símbolos especiais: ",", ":", ";", "."
8. Bloco de comandos delimitados por begin e end
9. Comentário de bloco com os delimitadores "/*" e "*/"
10. Lista de Palavras Reservadas:

Palavras Reservadas da Linguagem: ABSOLUTE ARRAY BEGIN CASE CHAR CONST DIV DO DOWTO ELSE END EXTERNAL FILE FOR FORWARD FUNC FUNCTION GOTO IF IMPLEMENTATION INTEGER INTERFACE INTERRUPT LABEL MAIN NIL NIT OF PACKED PROC PROGRAM REAL RECORD REPEAT SET SHL SHR STRING THEN TO TYPE UNIT UNTIL USES VAR WHILE WITH XOR**.**

2. Implementação
Observações:

- O compilador deverá ser feito em ambiente Windows, usando para tanto a linguagem de programação de familiaridade do aluno, porém escolha uma linguagem que permita construir uma interface amigável com o usuário.
- O analisador léxico deverá gerar como saída uma listagem mostrando o par *lexema* e *token* reconhecidos em todo o código fonte.
- O trabalho deve ser implementado em grupos de 2 alunos.
- O projeto deverá ser feito com base em um autômato finito determinístico, construído para a linguagem. O trabalho poderá ser feito em dupla, porém a avaliação deve ser individual, conforme participação e desenvoltura na implementação e apresentação do mesmo. O material produzido deverá ser entregue até a primeira aula após a realização da prova da I unidade.

1. Cada dupla deverá apresentar seu trabalho para a turma em data a ser marcada posteriormente.

Elabore autômatos finitos para os itens léxicos da linguagem especificada;

Quanto às estruturas de dados auxiliares, escolha uma das opções:

- Implemente uma função que constrói a tabela de palavras reservadas. Essa função será executada uma única vez no início da execução do compilador;
- Implemente a tabela de palavras reservadas como parte da tabela de símbolos. Nesse caso, a tabela de símbolos deverá ser devidamente inicializada no início da execução do compilador; Em ambos os casos, deve ser implementada a estrutura de dados completa (dados mais funções de manipulação).

Implemente o **analisador léxico** como uma função (pois depois será utilizado pelo analisador sintático). Sempre que chamado, ele retorna dois valores: uma cadeia lida e o token correspondente. Os erros léxicos devem ser detectados e tratados apropriadamente.

Deverão ser identificados os seguintes Símbolos da Linguagem:

- **Palavra Reservada**
- **Identificador** (Ex: x, variável, i, var2)
- **Número Inteiro** (Ex: 1, 13)
- **Número Real** (Ex: 1.33, 24.40e-04)
- **Operador aritmético** (“+”, “-, “*”, “/”, mod)
- **Operador relacional** (“=”, “>=”, “ >”, “<” ,“<=” ,“<>”)
- **Operador lógico** (and, or, not)
- **Símbolo Especial** (“=”, “(“, “)”, “,”, “;”, “:”)
- **Atribuição** (:=)
- **Fim** (.)

Deverão ser desconsiderados os caracteres não significativos: espaços, tabs, enters e **comentários de bloco** (/* qualquer coisa */). Então para não haver dúvida levantada em sala de aula, é preciso tratar os comentários de maneira explícita, ou seja, não podem aparecer na saída do programa porque não são tokens.
Qualquer outro símbolo deverá ser considerado desconhecido. Mari&

O trabalho será testado da seguinte maneira:

Entrada: um arquivo-fonte em formato texto (.txt) contendo o programa na linguagem especificada anteriormente

Saída: um arquivo em formato texto (.txt) contendo os tokens encontrados pelo analisador léxico. Deve-se criar um arquivo de saída com um par <cadeia, token> por linha.

Algumas decisões de projeto e de implementação que o grupo de trabalho deverá tomar:

- Vai existir um limite para o tamanho dos identificadores?
- Quais os tokens para as cadeias: <palavra_reservada, palavra_reservada> ou <palavra_reservada, t_palavra_reservada>? Não use códigos numéricos para os tokens.
- Como implementar a tabela de palavras reservadas/símbolos? Dependendo da opção escolhida acima, você deverá definir a estrutura de uma ou duas tabelas. Escolha a(s) estrutura(s) de dados e defina as funções de manipulação da(s) tabela(s). Lembre-se que a busca nessa(s) tabela(s) deve ser eficiente.
- As informações que ficarão na tabela de símbolos serão inseridas pelo analisador léxico ou sintático?
- Como serão tratados os erros léxicos? Especifique que estratégias de correção serão adotadas.

Um exemplo de um arquivo de entrada:

```
program teste;
var x,y: integer;
const pi := 3.1416;
/* inicio do programa */
begin
        read(x);
        if (x > y) then
```

```
                y := x ;
        else
                y := -x;
        writeln(x);
        write(y);
end.
```

Executando esta entrada, teremos como resposta:

```
program         PALAVRARESERVADA
teste           IDENTIFICADOR
;               SIMBOLOESPECIAL
var             PALAVRARESERVADA
x               IDENTIFICADOR
,               SIMBOLOESPECIAL
y               IDENTIFICADOR
:               SIMBOLOESPECIAL
integer         PALAVRARESERVADA
;               SIMBOLOESPECIAL
const           PALAVRARESERVADA
pi              IDENTIFICADOR
:=              ATRIBUIÇÃO
3.1416          NUMEROREAL
;               SIMBOLOESPECIAL
begin           PALAVRARESERVADA
read            IDENTIFICADOR
(               SIMBOLOESPECIAL
x               IDENTIFICADOR
)               SIMBOLOESPECIAL
;               SIMBOLOESPECIAL
if              PALAVRARESERVADA
(               SIMBOLOESPECIAL
x               IDENTIFICADOR
)               SIMBOLOESPECIAL
then            PALAVRARESERVADA
y               IDENTIFICADOR
:=              ATRIBUICAO
x               IDENTIFICADOR
else            PALAVRARESERVADA
y               IDENTIFICADOR
:=              ATRIBUICAO
-               OPERADOR
x               IDENTIFICADOR
;               SIMBOLOESPECIAL
writeln         IDENTIFICADOR
(               SIMBOLOESPECIAL
x               IDENTIFICADOR
)               SIMBOLOESPECIAL
;               SIMBOLOESPECIAL
write           IDENTIFICADOR
(               SIMBOLOESPECIAL
y               IDENTIFICADOR
)               SIMBOLOESPECIAL
;               SIMBOLOESPECIAL
end             PALAVRARESERVADA
```

. FIM

```
/* Outro exemplo de programa */
Program Piloto;
   /* declarações de variáveis e constantes globais */
     var cont, total: integer ;
         Nota1, Nota2, Media_das_medias, med: real;
/* Inicio do Programa */
begin
        media_das_medias := 0;
        writeln("******** ENTRADA DE DADOS ***************");
        writeln("Digite o total de alunos");
        read(total);
        for cont=1 to total do
        begin
                writeln("Digite os valores da primeira nota do aluno ", cont);
                read(Nota1);
                writeln("Digite os valores da segunda nota do aluno ", cont);
                read(Nota2);
                med := (Nota1+Nota2)/2.0;
                media_das_medias := media_das_medias + med;
                write("Media = ",med);
        end;
        write("Media Geral = ", Media_das_medias/total);
end.
```

Deverão ser entregues:

- Texto de documentação do programa, com explicações a respeito da instalação e execução do mesmo.
- Código fonte, ricamente comentado.
- Arquivos da linguagem utilizada (preparados para serem executados diretamente, sem necessidade de recompilação ou qualquer ambiente de desenvolvimento).
- Conjunto de, pelo menos, 5 arquivos de teste, contendo programas na linguagem, que permitam testar exaustivamente o analisador.

Critérios de avaliação:

Os trabalhos serão avaliados de acordo com os seguintes parâmetros:

- Funcionamento: o programa deve estar isento de erros.
- Organização do código: o código fonte deve estar estruturado de maneira adequada.
- Documentação: o programa deve estar adequadamente documentado.
- Interface: o programa deve ser fácil de ser usado e produzir mensagens de erro que esclareçam o usuário.
- O autômato deve estar escrito de forma adequada e correto.
- Não serão aceitos trabalhos com mais de 4 dias após a data de entrega dos mesmos, considerando uma redução de 1,0 a cada dia de atraso.

Trabalho prático da II unidade

Implementação do Analisador Sintático e Semântico - Valor 4,0

Criar, em dupla, um analisador sintático e semântico que reconheçam programas escritos na linguagem especificada segundo a gramática anterior. A dupla deverá utilizar a técnica SLR(1) para a implementação do analisador sintático para a linguagem especificada acima.

Algumas recomendações para o trabalho:

- Use o analisador léxico feito no primeiro trabalho (com as devidas correções)
- Identifique os **tokens** existentes nesta gramática (símbolos terminais e definições regulares), modificando o analisador léxico se necessário.

- Comece implementando um analisador sintático simples para um subconjunto da gramática (para reconhecer expressões válidas, por exemplo). Modifique a gramática (se necessário), gere a tabela de símbolos e implemente o analisador léxico para este subconjunto
- O analisador sintático deverá receber um arquivo com um código-fonte e devolver como saída "OK" se estiver tudo correto ou um conjunto de mensagens de erro caso contrário. Quando encontrar um erro sintático, é preciso imprimir uma mensagem de erro indicando o tipo de erro e o número da linha de ocorrência.

Deverão ser entregues:
- Documentação impressa do trabalho, com explicações a respeito da instalação e utilização do mesmo. Este texto deverá conter:
- Descrição do que foi feito e o do que *não* foi feito.
- Descrição do que está funcionando e do que *não* está funcionando.
- Comentários a respeito da linguagem, descrevendo suas limitações e incluindo eventuais modificações na gramática projetada inicialmente.
- Uma avaliação do grupo a respeito do trabalho.
- Código fonte, ricamente comentado.
- Arquivos da linguagem utilizada (preparados para serem executados diretamente, sem necessidade de recompilação ou qualquer ambiente de desenvolvimento).
- Conjunto de pelo menos 5 arquivos de teste, contendo programas na linguagem, que permitam testar exaustivamente o analisador.

Observações:
- Os trabalhos serão avaliados de acordo com o texto explicativo. Os códigos fontes e executáveis serão utilizados apenas para complemento da avaliação.
- Discrepâncias entre o texto e o trabalho entregue serão consideradas como tentativa de fraude.
- Não serão aceitos trabalhos com mais de 4 dias após a data de entrega dos mesmos, considerando uma redução de 10% a cada dia de atraso.
- Trabalhos que não possam ser executados terão nota zero.

Critérios de avaliação:
- Domínio dos apresentadores com relação ao conteúdo do trabalho.
- Qualidade do material utilizado na apresentação.
- Funcionamento: o programa deve estar isento de erros.
- Organização do código: o código fonte deve estar estruturado de maneira adequada.
- Documentação: o programa deve estar adequadamente documentado.
- Interface: o programa deve ser fácil de ser usado e produzir mensagens de erro que esclareçam o usuário.

ESPECIFICAÇÃO DA GRAMÁTICA DA LINGUAGEM:
Aqui você encontra a especificação da gramática da linguagem a ser usada pelo Analisador Sintático que será desenvolvido na disciplina Compiladores.

1. **REGRAS GRAMATICAIS**

PROG > "program" ident ";" BLOCO "."
BLOCO > { "var" { ident {"," ident}* ":" tipo ";"}+ }*
{"procedure" ident PALIST ";" BLOCO ";"}*
"begin" {COMANDO ";"}+ "end"
PALIST > ["(" { { ["var"] {ident || ","}+ ":" tipo } || ";" }+ ")"]
COMANDO > variavel ":=" EXPR | proc ["(" {EXPR || ","}+ ")"] |
"if" EXPR "then" COMANDO ["else" COMANDO] |
"while" EXPR "do" COMANDO |

“begin” {COMANDO “;”}+ “end”
EXPR > SIEXPR [(“=” | “<” | “>” | “<>” | “<=” | “>=”) SIEXPR]
SIEXPR > [“+” | “”] {TERMO || “+” | “”}+
TERMO > {FATOR || “*” | “div” | “mod”}+
FATOR > numero | variável | “(” EXPR “)”

2. SÍMBOLOS TERMINAIS
Item léxico: {“program”, “;”, “.”, “var”, “,”, “.”, “begin”, “end”}
Os itens léxicos não têm acentuação.
Classe léxica: {ident, tipo, proc, número, variável}.

3. SÍMBOLOS NÃO TERMINAIS: {PROG, BLOCO, COMANDO, PALIST, EXPR,
SIEXPR, TERMO,
FATOR}, conforme descrição a seguir.
PROG – Símbolo inicial da gramática que representa um programa;
BLOCO – Representa a definição de variáveis e procedimentos e o corpo do programa;
COMANDO – Representa um comando;
PALIST – Lista de parâmetros;
EXPR – Expressão;
SIEXPR – Expressão simples;
TERMO – Termo de uma expressão;
FATOR – Um fator de uma expressão.

4. METASÍMBOLOS
(Símbolos da linguagem usada para especificar a gramática):
[A] – Os colchetes denotam opcionalidade, ou seja, a estrutura A é opcional – pode aparecer ou não. Representa o conjunto { , A}.
{B}* Representa a concatenação de B, ou seja, o conjunto { , B, BB, BBB, ...}. A estrutura B se repete 0 (zero) ou mais vezes.
{C || D}+ Representa a intercalação, ou seja, o conjunto {C, CDC, CDCDC, ...}. Sempre que a estrutura C se repetir ela será precedida pela estrutura D.
E | F – É a alternação. Representa a estrutura E ou a estrutura F, ou seja o conjunto {E, F}.