A proposta desta atividade é a implementação das etapas *front-end* do projeto de uma linguagem de programação, ou seja, as primeiras etapas da construção do compilador de uma linguagem específica.

Você construirá o *front-end* da linguagem MiniPascal (que se trata de uma simplificação da linguagem Pascal) e deve utilizar a linguagem C para a implementação.

Etapas:

1. Gramática livre de contexto e autômatos finitos
2. Analisador léxico
3. Analisador sintático
4. Analisador semântico

**OBS1:** todas as etapas devem ser entregues na data determinada.

**OBS2:** cada entrega deve ser composta por 2 arquivos: código e documentação (ou simplesmente código muito bem comentado).

**OBS3:** a não entrega de uma etapa zera o projeto.

**ETAPA#1 – Expressões Regulares, Gramática Livre de Contexto e Autômatos Finitos**

* A partir das definições da linguagem específica e da sugestão de sintaxe (final do arquivo), apresentar as expressões regulares e construir a gramática livre de contexto correspondentes.
* Construir os autômatos para cada classe de elementos do vocabulário terminal da linguagem: letras, dígitos, números, identificadores, operadores, palavras reservadas, pontuação, comentários.
* Testar cada classe gerada.
* Juntar todos os autômatos em um único para que represente todos os elementos.

**OBS4:** encaminhar o autômato completo e não separadamente para cada padrão definido.

**OBS5:** utilizar a ferramenta JFLAP ([www.jflap.org](http://www.jflap.org)) para a construção dos autômatos.

**ETAPA#2 – Analisador léxico**

* Desenvolver um programa que realize a análise léxica.
* A partir de um arquivo contendo um código (programa) em MiniPascal, o analisador léxico deverá produzir um arquivo com os *tokens* identificados e seus valores correspondentes (se necessário).
* Considerar que todos os *tokens* estão separados por um espaço.
* Cada linha do arquivo de saída deve conter um *token* reconhecido.
* Fazer uso do que foi implementado na **Etapa#1.**
* O analisador léxico deve ser capaz de lidar com os erros léxicos encontrados no programa. A mensagem de erro, “ERRO LÉXICO” juntamente com a sequência lexicamente errada, devem ser apresentadas e todo o processo finalizado.

**ETAPA#3 – Analisador sintático**

* Construir um programa que realize a análise sintática.
* A partir do arquivo com os *tokens* gerados pelo analisador léxico (**Etapa#2**), realizar a análise ascendente ou descendente com base na gramática da linguagem.
* O analisador sintático deve ser capaz de lidar com erros sintáticos encontrados no programa. A mensagem de erro, “ERRO SINTÁTICO” juntamente com o *token* incorreto, devem ser apresentados e todo o processo finalizado.
* É necessário entregar os analisadores léxico e sintático funcionando em conjunto.

**ETAPA#4 – Analisador semântico**

* Deve-se construir e/ou atualizar a tabela de símbolos e fazer a checagem de tipos.
* É necessário entregar todos os analisadores desenvolvidos funcionando em conjunto.

**CRITÉRIO DE AVALIAÇÃO**

* **Etapa#1 – 15%**
* **Etapa#2 – 15%**
* **Etapa#3 – 15%**
* **Etapa#4 – 15%**

**OBS6:** o desenvolvimento de cada ETAPA deve ser explicado e, se preciso for, exemplificado.

* Documentação interna: comentários claros e objetivos incluídos no código fonte do programa

**DATAS IMPORTANTES**

* **Etapa#1 – 29/AGO/2017**
* **Etapa#2 – 19/SET/2017**
* **Etapa#3 – 17/OUT/2017**
* **Etapa#4 – 14/NOV/2017**

**GRUPOS**

* Grupos de no máximo 4 alunos. Trabalhos individuais não serão aceitos.

**MINIPASCAL**

MiniPascal é um subconjunto da linguagem Pascal. Como é um subconjunto, todo programa MiniPascal é um programa Pascal válido que pode ser executado.

MiniPascal restringe a linguagem Pascal para ter apenas inteiros e booleanos. Os comandos permitidos são aqueles que podem ser usados com os tipos de dados mencionados. Os literais não são considerados para manipulação, apenas nos *prints* para mensagem de interação com o usuário (em Pascal o literal é colocado entre aspas duplas).

MiniPascal é uma linguagem *case sensitive*.

**Exemplo de código correto**

program correto ;

var a, b, c : integer ;

var d, e, f : boolean ;

procedure proc(var a1 : integer ) ;

var a, b, c : integer ;

var d, e, f : boolean ;

begin

a := 1 ;

if ( a < 1 ) then

a := 12 ;

end ;

begin

a := 2 ;

b := 10 ;

c := 11 ;

a := b + c ;

d := true ;

e := false ;

f := true ;

write ( b ) ;

if ( d ) then

begin

a := 20 ;

b := 10 \* c ;

c := a div b ;

end ;

while ( a > 1 ) do

begin

if ( b > 10 ) then

b := 2 ;

a := a - 1 ;

end ;

end .

Um programa MiniPascal deverá estar codificado em um único arquivo fonte, sem fazer referências a detalhes externos a ele.

**ESPECIFICAÇÃO LÉXICA**

* **Comentários**

Aparecem delimitados por “/\*” e “\*/”. Tudo que segue os símbolos “/\*” é ignorado pelo MiniPascal até que se encontre “\*/”.

/\* comentário \*/

* **Identificadores (variáveis)**

Podem ser criados com caracteres alfanuméricos e o caractere ‘\_’. Um identificador não pode começar com um número, ou seja, é iniciado por uma letra ou ‘\_’ e é seguido de zero ou mais letras, dígitos ou ‘\_’

Value

company\_number

12Val /\* inválido \*/

Ex$ /\* inválido \*/

company-name /\* inválido \*/

* **Tipos de dados**

Numerais : Inteiros (tipo integer)

Booleano : False | True (tipo boolean)

* **Operadores**

+ - div \* and or not

< > <> <= >= :=

* **Delimitadores**

( ) , : . ;

* **Palavras Reservadas**

**program begin end procedure**

**if then else while**

**do and or not**

**var integer boolean true**

**false write**

**SINTAXE**

1. <programa> ::=

**program** <identificador> **;**

<bloco>**.**

2. <bloco> ::=

[<parte de declarações de variáveis>]

[<parte de declarações de sub-rotinas>]

<comando composto>

**Declarações**

3. <parte de declarações de variáveis> ::=

**var** <declaração de variáveis>

{**;** <declaração de variáveis>}**;**

4. <declaração de variáveis>::=

<lista de identificadores> **:** <tipo>

5. <lista de identificadores> ::=

<identificador> {**,** <identificador>}

6. <parte de declarações de subrotinas> ::=

{declaração de procedimento> **;**}

7. <declaração de procedimento> ::=

**procedure** <identificador> [<parâmetros formais>] **;** <bloco>

8. <parâmetros formais> ::=

**(** <seção de parâmetros formais> { **;** <seção de parâmetros formais>} **)**

9. <seção de parâmetros formais> ::=

[**var**] <lista de identificadores> **:** <tipo>

10. <tipo> ::=

**integer** | **boolean**

**OBS**: Todos os identificadores devem ser declarados antes de serem utilizados.

**Comandos**

11. <comando composto ::=

**begin** <comando> { **;** <comando>} **end**

12. <comando> ::=

<atribuição>

| <chamada de procedimento>

| <comando composto>

| <comando condicional 1>

| <comando repetitivo 1>

| **write**( <identiicador> )

13. <atribuição>::=

<variável> **:=** <expressão>

14. <chamada de procedimento> ::=

<identificador> [ **(** <lista de parâmetros> **)** ]

15. <lista de parâmetros> ::=

[ ( <identificador> | <numero> | <bool> )

{, ( <identificador> | <numero> | <bool> ) } ]

16. <comando condicional 1> ::=

**if** (<expressão>) **then** <comando>

[**else** <comando>]

17. <comando repetitivo 1> ::=

**while** (<expressão>) **do** <comando>

**Expressões**

18. <expressão> ::=

<expressão simples> [<relação> <expressão simples>]

19. <relação> ::=

**=** | **<>** | **<** | **<=** | **>=** | **>**

20. <expressão simples> ::=

[**+** | **-**] <termo> {(**+** | **-**) <termo>}

21. <termo> ::=

<fator> {(**\*** | **div**) <fator> }

22. <fator> ::=

<variavel>

| <número>

| <bool>

| **(** <expressão simles> **)**

23. <variável> ::=

<identificador>

**Números e Identificadores**

24. <bool> ::=

**true** | **false**

25. <número> ::=

<dígito> {<dígito>}

26. <dígito> ::=

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

27. <identificador> ::=

<letra> {<letra> | <dígito>}

28. <letra> ::=

\_ | a | b | ... | z | A | B | ... | Z

**EBNF:**

{α} = repetição da cadeia a zero ou mais vezes

α | β = α ou β devem ser escolhidos

Não terminais aparecem entre < e >.