



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* de uma linguagem simplificada aqui chamada de **AlgC** (que se trata de uma adaptação simplificada da linguagem C) e deve utilizar a linguagem C para a implementação do compilador.

Etapas:

- (1) Autômato finito
- (2) Analisador léxico
- (3) Analisador sintático
- (4) Analisador semântico

**OBS<sub>1</sub>:** todas as etapas devem ser entregues na data determinada.

**OBS<sub>2</sub>:** cada entrega deve ser composta por 2 arquivos: código e documentação (ou, simplesmente, o código muito bem comentado).

#### ETAPA#1 – 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.

**OBS<sub>4</sub>:**encaminhar o autômato completo e não separadamente para cada padrão definido.

**OBS<sub>5</sub>:**utilizar a ferramenta JFLAP **versão 7.0** ([www.jflap.org](http://www.jflap.org)) para a construção dos autômatos.

#### ETAPA#2 – Analisador léxico

- Desenvolver um programa que realize a análise léxica.
- O analisador léxico deverá ser fiel ao afd resultante da ETAPA#1 e codificado segundo a estratégia baseada no uso de rótulos e comandos **goto**.
- A partir de um arquivo contendo um código (programa) em AlgC, 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 **tokens** reconhecido.
- 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.
- Usar a sequência de tokens retornada pelo analisador Léxico (um de cada vez) em um Analisador Sintático Recursivo Descendente baseado na gramática da linguagem.
- Serão permitidos eventuais ajustes na gramática proposta (para adequá-la para um Analisador Sintático Recursivo Descendente).
- Todos os ajustes feitos na gramática deverão ser publicados em um documento que fará parte da entrega desta ETAPA.
- 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%

**OBS<sub>6</sub>:** 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

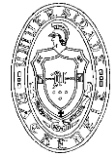
### DATAS IMPORTANTES

- Etapa#1 – 10/SET/2018
- Etapa#2 – 24/SET/2018
- Etapa#3 – 22/OUT/2018
- Etapa#4 – 19/NOV/2018

**GRUPOS** de no máximo 4 alunos. Trabalhos individuais serão aceitos somente se tratarem, adicionalmente, do comando **switch-case**, que não é descrito neste documento.

### OBSERVAÇÕES COMPLEMENTARES

- O trabalho deverá ser feito na linguagem C (padrão ANSI).
- Os trabalhos entregues por grupos deverão ser publicados no Moodle por um único aluno do grupo.
- Os trabalhos deverão estar documentados com as identificações completas de todos os membros do grupo.
- Os programas entregues deverão executar no DEV C++ para Windows. Para isso, as entregas envolvendo código em C deverão publicar uma pasta “*zipada*” contendo todo o projeto (em particular, deverá conter o arquivo de projeto de DEV C++).
- Não é permitido o uso de nenhuma biblioteca externa que não esteja disponível na instalação básica do DEV C++.
- **Projetos que não executarem no DEV C++ para Windows receberão nota ZERO.**



## AlgC

AlgC descreve um subconjunto de uma linguagem estruturada genérica. Por ser um subconjunto, todo programa AlgC pode ser facilmente compreendido, bastando que o aluno tenha conhecimentos básicos de linguagens de programação.

AlgC trata apenas de valores 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.

AlgC é uma linguagem *case sensitive*.

### Exemplo de código correto

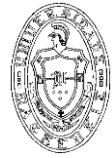
```
program _correto {
    int _a, _b, _c ;
    bool _d, _e, _f ;

    void _proc(int _arg) {
        int _a ;
        _a= 1;
        if (_a<1) {
            _a= 12;
        }
    }

    _a= 2;
    _b= 10;
    _c= 11;
    _a= _b+_c;
    _d= true;
    _e= false;
    _f= true;

    print(_b);

    if (_d) {
        _a= 20;
        _b= 10 * _c;
        _c= _a / _b;
    }
    while (_a>1) {
        if (_b>10) {
            _b= 2;
            _a= _a-1;
        } else {
            _a= _a-1;
        }
    }
}
```



Um programa em AlgC 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 MiniAlg até que se encontre “\*/”.

```
/* comentário */
```

### ▪ Identificadores (variáveis)

Podem ser criados somente com letras de **a** até **z** (maiúsculas ou minúsculas) e devem, obrigatoriamente, começar com o caractere **\_** (“underline”).

```
_Value  
_companyNumber  
  
_12Val          /* inválido */  
Val12           /* inválido */  
_Ex$            /* inválido */  
company-name    /* inválido */  
_company_name   /* inválido */
```

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

### ▪ Tipos de dados

Numerais : Inteiros (tipo **int**)

Booleano : **false** | **true** (tipo **bool**)

### ▪ Operadores

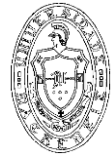
**+**   **-**   **/**   **\***   **<**   **>**   **!=**   **<=**   **>=**   **==**   **=**

### ▪ Delimitadores

**(**   **)**   **,**   **;**   **{**   **}**

### ▪ PalavrasReservadas

<b>program</b>	<b>if</b>	<b>else</b>
<b>void</b>	<b>true</b>	<b>false</b>
<b>int</b>	<b>bool</b>	<b>while</b>
<b>print</b>		



## SINTAXE

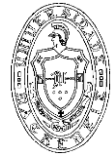
1.  $\langle \text{programa} \rangle ::= \text{program } \langle \text{identificador} \rangle \{ \langle \text{bloco} \rangle \}$
2.  $\langle \text{bloco} \rangle ::= [ \langle \text{parte declarações de variáveis} \rangle ]$   
 $[ \langle \text{parte declarações de funções} \rangle ]$   
 $\langle \text{comando composto} \rangle$

## Declarações

3.  $\langle \text{parte declarações de variáveis} \rangle ::= \langle \text{declaração de variáveis} \rangle \{ \langle \text{declaração de variáveis} \rangle \}$
4.  $\langle \text{declaração de variáveis} \rangle ::= ( \text{int} \mid \text{bool} ) \langle \text{lista de identificadores} \rangle ;$
5.  $\langle \text{lista de identificadores} \rangle ::= \langle \text{identificador} \rangle [ , \langle \text{identificador} \rangle ]$
6.  $\langle \text{parte declarações de funções} \rangle ::= \{ \langle \text{declaração de função} \rangle ; \}$
7.  $\langle \text{declaração de função} \rangle ::=$   
 $\text{void } \langle \text{identificador} \rangle ( [ \langle \text{parâmetros formais} \rangle ] ) \{ \langle \text{bloco} \rangle \}$
8.  $\langle \text{parâmetros formais} \rangle ::= \langle \text{parâmetro formal} \rangle \{ , \langle \text{parâmetro formal} \rangle \}$
9.  $\langle \text{parâmetro formal} \rangle ::= ( \text{int} \mid \text{bool} ) \langle \text{identificador} \rangle$

## Comandos

10.  $\langle \text{comando composto} \rangle ::= \langle \text{comando} \rangle ; [ \langle \text{comando} \rangle ; ]$
11.  $\langle \text{comando} \rangle ::= \langle \text{atribuição} \rangle$   
 $\mid \langle \text{chamada de procedimento} \rangle$   
 $\mid \langle \text{comando condicional} \rangle$   
 $\mid \langle \text{comando repetitivo} \rangle$   
 $\mid \text{print } ( \langle \text{identificador} \rangle ) ;$
12.  $\langle \text{atribuição} \rangle ::= \langle \text{variável} \rangle = \langle \text{expressão} \rangle ;$
13.  $\langle \text{chamada de procedimento} \rangle ::= \langle \text{identificador} \rangle [ ( \langle \text{lista de parâmetros} \rangle ) ] ;$
14.  $\langle \text{lista de parâmetros} \rangle ::=$   
 $[ ( \langle \text{identificador} \rangle \mid \langle \text{número} \rangle \mid \langle \text{bool} \rangle )$   
 $\{ , ( \langle \text{identificador} \rangle \mid \langle \text{número} \rangle \mid \langle \text{bool} \rangle ) \} ]$
15.  $\langle \text{comando condicional} \rangle ::=$   
 $\text{if } ( \langle \text{expressão} \rangle ) \{ \langle \text{comando composto} \rangle \} [ \text{else } \{ \langle \text{comando composto} \rangle \} ]$



16.  $\langle \text{comando repetitivo} \rangle ::= \text{while } ( \langle \text{expressão} \rangle ) \{ \langle \text{comando composto} \rangle \}$

### Expressões

17.  $\langle \text{expressão} \rangle ::= \langle \text{expressão simples} \rangle [ \langle \text{relação} \rangle \langle \text{expressão simples} \rangle ]$

18.  $\langle \text{relação} \rangle ::= == \mid != \mid < \mid <= \mid >= \mid >$

19.  $\langle \text{expressão simples} \rangle ::= [ + \mid - ] \langle \text{termo} \rangle \{ [ + \mid - ] \langle \text{termo} \rangle \}$

20.  $\langle \text{termo} \rangle ::= \langle \text{fator} \rangle \{ ( * \mid / ) \langle \text{fator} \rangle \}$

21.  $\langle \text{fator} \rangle ::=$   
     $\langle \text{variável} \rangle$   
     $\mid \langle \text{número} \rangle$   
     $\mid \langle \text{bool} \rangle$   
     $\mid ( \langle \text{expressão simples} \rangle )$

22.  $\langle \text{variável} \rangle ::= \langle \text{identificador} \rangle$

### Números e Identificadores

23.  $\langle \text{bool} \rangle ::= \text{true} \mid \text{false}$

24.  $\langle \text{número} \rangle ::= \langle \text{dígito} \rangle \{ \langle \text{dígito} \rangle \}$

25.  $\langle \text{dígito} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

26.  $\langle \text{identificador} \rangle ::= \_ \langle \text{letra} \rangle \{ \langle \text{letra} \rangle \}$

27.  $\langle \text{letra} \rangle ::= a \mid b \mid \dots \mid z \mid A \mid B \mid \dots \mid Z$

### EBNF:

$\{ \alpha \}$  = repetição da cadeia  $\alpha$  a zero ou mais vezes

$[ \alpha ]$  = cadeia  $\alpha$  é opcional

$( \alpha \mid \beta )$  =  $\alpha \mid \beta$  =  $\alpha$  ou  $\beta$  deve ser escolhido

Não terminais aparecem entre  $\langle$  e  $\rangle$ .