Compiladores – *Implementação – 2017/2*

Regras

- Palavras Reservadas:
    - programa
    - leia
    - escreva
    - se
    - senao
    - para
    - inteiro
    - caractere
    - real
    - fim
    - Importante leia <> Leia <> LEIA, então verifique exatamente como descrito (letras minúsculas)

1. O arquivo começa com a palavra reservada *programa* e termina com a palavra reservada *fim*

2. Declaração de variáveis
    2.1. Sempre deve conter o tipo de dado (inteiro, caractere (e seu tamanho – limitador de tamanho "[]") ou real(e as especificações de casas decimais - e seu tamanho – limitador de tamanho "[]"));
    2.2. A declaração de variável poderá ser feita somente no início do código especificando o tipo de dado da variável.
    2.3. Todas as variáveis precisam do marcador #. Após o "#" deve-se ter um(01) símbolo de a...z (minúsculo) e após se necessário pode ser inserido qualquer símbolo de a..z ou A...Z ou 0...9.
    2.4. Nenhum caractere especial será aceito na formação das variáveis.
    2.5. A linha deve ser finalizada com ponto e vírgula;
    2.6. Poderá em uma linha haver mais de uma variável declarada para o mesmo tipo, desde que separadas por vírgula;
    2.7. O separador de casas decimais será o símbolo "." (ponto), considerando-se que deve explicitar a quantidade antes e depois da vírgula, separadamente – considerando valores separados.
    2.8. Atribui-se valores a uma variável utilizando o símbolo "=" (igual e somente um). Na sua declaração ou após.

2.9. As atribuições de variáveis devem obedecer ao escopo da variável, para caractere utilizar a atribuição com aspas duplas, para inteiro considerar somente o número inteiro, e para real considerar casas antes e após o ponto, conforme descrito na declaração;
2.10. Em atribuições podem ser feitos tanto com valor, quanto com outra variável.
2.11. Na declaração é possível atribuir somente valores;
2.12. Poderão haver operações matemáticas no decorrer do código – Considere + pra soma, * para multiplicação, - para subtração, / para divisão e ^ para exponenciação. Poderão ser "[]" utilizados para delimitar prioridades, caso não utilize considerar as regras de matemática;

Exemplos:

```
inteiro #c;
inteiro #b, #a;

#a = #b + [[5*8]/2];
#b=4;

real #a[6.2];

caractere #nome[5]=”Aline”;
```

3. Leitura
3.1. O comando de leitura – leia – poderá ler mais de uma variável, porém as variáveis devem ser separadas por vírgula e declaradas anteriormente;
3.2. Não pode ser feita declarações dentro da estrutura de leitura.
3.3. Haverá sempre um duplo balanceamento utilizando os parênteses.
3.4. A linha deve ser finalizada com ponto e vírgula;
3.5. Podem ser lidas variáveis de tipos diferentes no mesmo comando, desde que declaradas anteriormente;

Exemplos:

```
inteiro #a, #b;
caractere #nome[30];
leia( #a,#b);
leia(#a);
leia(#b);
leia(#nome, #b);
```

4. Escrita
   4.1. O comando de escrita – escreva – poderá escrever mais de uma variável;
   4.2. Poderá mesclar texto e variável, desde que tenha o símbolo “,” que deve ser utilizado após (e/ou antes) das aspas duplas do texto – que deverá ter duplo balanceamento.
   4.3. Podem ser escritas variáveis de tipos diferentes no mesmo comando, desde que declaradas anteriormente;
   4.4. Os textos que precisarem ser escritos no comando escreva devem estar dentro das aspas duplas.
   4.5. Variáveis estarão fora das aspas duplas.
   4.6. Se houver escrita de mais de uma variável deverá ser separada com “,” e já devem ter sido declaradas anteriormente. Observando que irá agrupar os conteúdos.
   4.7. Não pode ser feita declarações dentro da estrutura de escrita.
   4.8. Haverá sempre um duplo balanceamento utilizando os parênteses para o escreva e aspas duplas para texto.

Exemplos:

```
escreva (“Textos.sdskfhdfhgasdfajdh”);
escreva (#a, #b);
escreva (#a);
escreva (#a, “texto”,#b);
escreva (“Texto”,#a, “Texto”,#a,#b)
escreva (“Texto”,#a, “ , ”,#a,“ , ”+#b)
```

5. Se
   5.1. O comando de teste - se - deve conter obrigatório um teste e uma condição de verdadeiro, podendo ou não conter um comando de falso – senao.
   5.2. Nos comandos de verdadeiro **e/ou** falso podem conter várias linhas (considere a necessidade de abrir e fechar o bloco com “{}”, **somente para mais de uma linha**), e pode conter qualquer estrutura da linguagem, exceto declaração de variáveis.
   5.3. A linha do teste (se) não conterá finalização de linha (ponto e vírgula) as demais – condição verdadeira **e/ou** falsa - devem conter a finalização de linha com ponto e vírgula.
   5.4. O teste poderá ser variável com variável, variável com texto/número ou texto/número com variável – entenda que a palavra texto utilizada anteriormente também pode-se tratar de um número decimal ou inteiro, porém com as aspas duplas.
   5.5. Os seguintes operadores serão válidos:
      5.5.1. Para texto “==” (igual) ou “<>” (diferente);
      5.5.2. Para números: “==” (igual); “<>” (diferente); “<” (menor) ou “<=” (menor ou igual); “>” (maior) ou “>=” (maior ou igual);
      5.5.3. Não serão válidos os operadores invertidos =<, => ou ><, !=, <<, >>.

5.6. Se o teste feito for com texto (caractere) deverá conter aspas duplas respeitando o duplo balanceamento, e sempre deve haver o duplo balanceamento de parênteses.
5.7. Não haverá condições duplas, porém pode haver testes encadeados;

Exemplos:

```
se( #a==1)
    linha 1;
senão {
    Linha 1;
    Linha 2;}
```

```
se( #a==1)
    se (#b<>1)
            linha 1;
senao
    se( #a<1)
            se(#a<=5)
                linha 1;
        senão Linha 1 ;
```

6. Para
6.1. O laco de repetição – for - terá a seguinte estrutura para(x1;x2;x3), onde x1 – refere-se à atribuição de valor inicial da variável; x2 refere-se ao teste que deve ser feito a cada interação; x3 será o incremento ou decremento;
6.2. Pode-se iniciar uma variável com um valor fixo, ou com o conteúdo de outra variável (*comando de atribuição*), ou ainda não iniciá-la. No primeiro parâmetro poderá ser utilizado somente números inteiros. As variáveis já devem ter sido declaradas anteriormente.
6.3. No segundo parâmetro, o teste, deve-se comparar a variável inicializada anteriormente com número fixo, ou outra variável, podendo utilizar os operadores relacionais <, >, <= ou >= (observe as regras de formação destes operadores) ;
6.4. Para blocos de mais de uma linha deve-se utilizar "{}" para delimitar o início e fim;

Exemplos:

```
para (#a=#b;#a<=10;#a++)  {
    escreva (#a);
    escreva ("Texto");
    }
```

```
.
.
 #a=#b
.
.
.
para (;#a<=10;++#a)
    escreva (#a);
```

6.5. Qualquer comando pode ser executado dentro do laço de repetição, inclusive outro laço;
6.6. No terceiro parâmetro, pode aparecer um incremento ou decremento com a variável, ou mesmo uma operação (adição, subtração, multiplicação ou divisão);

Exemplos:

```
para (#a=#b;#a<=10;#a++)
para (#a=#b;#a<=10;++#a)
para (#a=#b;#a<=10;#a=#a*2)
```

7. Espaços
   7.1. *Poderá* aparecer entre uma palavra reservada e o próximo comando (seja ele qual for); ex: leia (“
   7.2. *Poderá* aparecer entre a vírgula e uma variável, ou a variável e uma vírgula, mas não irá interferir – seja na leitura, escrita ou declaração de variáveis;
   7.3. ***Não pode*** aparecer entre os comandos de teste com operados relacionais duplicados (<=, >=, == ou <>)
   7.4. **Não pode** “quebrar” a sequência de uma palavra reservada ou variável.
   7.5. Nas linhas que não possuem finalização de linhas – ponto e vírgula, entenda que ao(s) espaço(s) ***poderá*** acrescentar uma quebra de linha – **ou** troca-se o espaço por quebra de linha;
   7.6. Nas linhas que possuem finalização de linha entende-se que após a finalização haverá uma quebra de linha.

8. Memória utilizada
   8.1. O software deve ser capaz de fazer alocações dinâmica na memória, e ainda liberar a memória alocada, quando não está mais sendo utilizada e/ou realocar a memória se for o caso (a critério). E se não houver memória emitir a mensagem de **ERRO** “Memória Insuficiente”. E ainda ao final liberar toda a memória alocada;
   8.2. Apresentar o valor máximo de memória utilizada.
   8.3. A Memória disponível não poderá ultrapassar 350 MB.

Analise léxica:

- Você deve percorrer o código e fazer a análise léxica das palavras reservadas e variáveis da linguagem existentes no código que está sendo verificado, mostrar **erro** (mostrar o número da linha ou a própria linha) se em qualquer um deles apresentar problemas quanto a formação;
- Mostrar ao final a tabela de símbolos construída e a memória utilizada.

Analise Sintática:

- Após percorrer o código fazendo a análise léxica, verifique todas as palavras reservadas do sistema quanto à estrutura de escrita da função, duplos balanceamentos, os operadores lógicos, relacionais quando se fizerem necessário e mostrar os **erros** quando necessário (mostrar o número da linha ou a própria linha).
- Mostrar ao final a tabela de símbolos construída e a memória utilizada.

Analise semântica:

- A análise semântica **pode** acontecer simultaneamente a análise sintática, e deve apenas mostrar o ***alerta****(não finaliza o programa)* (mostrar o número da linha ou a própria linha) da execução, se houver.
- Mostrar ao final a tabela de símbolos construída e a memória utilizada.

- **1ª Entrega 13/11 – Analise Léxica**
- **2ª Entrega 11/12 – Analise Léxica, Sintática e Semântica**

- **Delay de 6 horas.**