Aula 1 - Conceito de Compiladores

O que é compilador?

O compilador é um **software que analisa o código verificando se tem erros**, e caso não tenha ele **faz a tradução (conversão) do código para linguagem de máquina criando um executável**.

Como funciona?

O compilador **analisa o código e através de um autômato** que faz a **análise e extração da palavra**, com isso **faz um select para comparar a palavra do código com o dicionário** do banco de dados.

Autômatos

São **tipos especiais de grafo**. Faz **o reconhecimento de vocábulos dentro de um texto extrai e identifica**. **EX:** Corretor Automático.

Tokens

São o **conjunto de palavras, símbolos e números**.

Processo de Compilação

1 - Análise Léxica

Faz a varredura no programa em busca de tokens inválidos.

Ex: imt x, f?oat → erro léxico.

2 - Análise Sintática

Verifica se a sequência de tokens está correta. São gabaritos sintáticos - São árvores binárias que verificam se cada posição bate ou não.

Ex: for(i=0; i<n; i++(→ erro sintático.

3 -Análise Semântica

Sem erros léxicos e sintáticos verifica o sentido de cada uma das sentenças.
Ex:
float media;
media = Math.Pow(aux,0.25); → erro semântico.

4 -Tradução

Sem nenhum tipo de erro então o programa é traduzido para linguagem de máquina.

Aula 2 - Arquivos Binários

Arquivo: Definição

Um arquivo é um conjunto de dados (bytes) armazenados em uma mídia, normalmente um disco.

Manipulando Arquivos

Leitura :

- O arquivo tem de existir.
- Necessário conhecer seu endereço: *lado,trilha e setor*.
- Descobrir o endereço do arquivo.
- Necessário abrir o arquivo antes de lê-lo.

EX: FileStream infile;
infile = new System.IO.FileStream("d:teste.txt",
System.IO.FileMode.Open,
System.IO.FileAccess.Read);

Gravar:

- O arquivo tem de não existir.
- Saber o endereço do local de gravação.
- Descobrir um endereço disponível onde possa ser gravado.
- Antes de ser gravado o arquivo deve ser aberto.

```
FileStream  outfile;
outfile= new System.IO.FileStream("d:teste2.txt",
                                  System.IO.FileMode.Create,
                                  System.IO.FileAccess.Write);
```

Lendo / Gravando um arquivo

Quando se abre um arquivo imediatamente nos posicionamos sobre o primeiro byte desse arquivo. Ao lermos ou gravarmos automaticamente nos posicionamos ao byte seguinte.

```
tam= (int) infile.Length;

    for (int i = 0; i < tam; ++i)
    {
        x = (byte) infile.ReadByte();
        outfile.WriteByte(x);
    }
```

Fechando um arquivo

Quando abrimos um arquivo vários componentes do SO são requisitados como ponteiros designados, **áreas de memória alocados (buffer)**. Ao terminar o arquivo **precisamos devolver ao sistema operacional os recursos**, **fechando o arquivo**.

EX: infile.close();

NOTA:

```
   iofile = new System.IO.FileStream("teste.txt",
                                     System.IO.FileMode.Open,
                                     System.IO.FileAccess.ReadWrite);

tam = (int)iofile.Length;
        for (int i = 0; i < tam; ++i)
        {
           x = (char)(iofile.ReadByte()+5);
           --iofile.Position; // para poder gravar o byte na posição certa.
position(ponteiro)
           iofile.WriteByte((byte)x);
        }
        iofile.Close();
        Console.ReadKey();
```

- No caso, *terá que subtrair o ponteiro* **--iofileposition** pois, ao ler **o primeiro caracter do arquivo automaticamente ele já é incrementado para o próximo, ou seja, pegaria o valor do segundo caracter e gravaria no lugar do primeiro.**

- **Bullet Points:** É necessário abrir o arquivo para mostrar ao sistema operacional seu **lado, trilha e setor**.

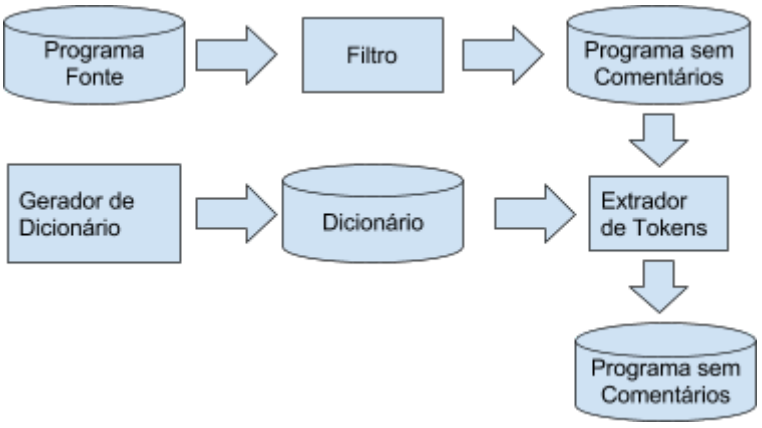
TP - Contar os Tokens e tirar os comentários.

```
FileStream infile, outfile;
int tam, contLetter = 0, contDigit = 0, contCaracter = 0, i = 0;
char x, aux = '#';
infile = new System.IO.FileStream("Teste.txt",
                                  System.IO.FileMode.Open,
                                   System.IO.FileAccess.Read);

outfile = new System.IO.FileStream("Copia.txt",
                                     System.IO.FileMode.Create,
                                     System.IO.FileAccess.Write);

tam = (int)infile.Length;
while (i < tam)
```

```
        {
        x = (char)infile.ReadByte();
        if (x == aux)
        {       i++;
               do
               {
               i++;
               x = (char)infile.ReadByte();
               }
               while (x != aux);
               x = (char)infile.ReadByte();
          }
        outfile.WriteByte((byte)char.ToUpper(x));

        if (char.IsLetter(x))
               contLetter++;
        else
               if (char.IsDigit(x))
               contDigit++;
               else
               contCaracter++;
        i++;
        }
        Console.WriteLine("A quantidade de letras é: " + contLetter);
        Console.WriteLine("A quantidade de dígitos é: " + contDigit);
      Console.WriteLine("A quantidade de outros caracteres é: " + contCaracter);
        infile.Close();
        outfile.Close();

        Console.ReadKey();
}
```

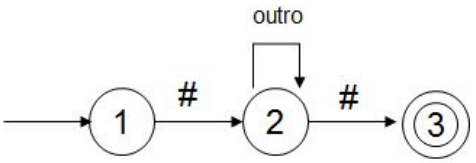
Aula 3 - Autômatos Finitos

O Analisor Léxico



Exemplo de Grafo Autômato Finito

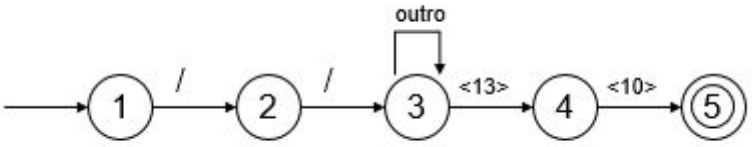
1: ***Encontrar um #***
2: ***Ler caracteres um-a-um até encontrar um #***
3: ***Fim do comentário,***



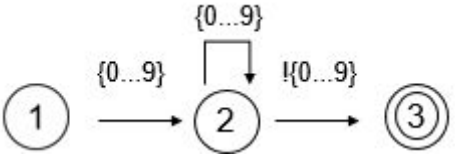
O que são Autômatos Finitos (Máquina de estado Finito)?

- Forma matemática (grafos) de descrever tipos particulares de algoritmo.
- No caso, são utilizados para **descrever o processo de reconhecimento de padrões em cadeia de caracteres**.

Autômato Para reconhecimento de comentário //



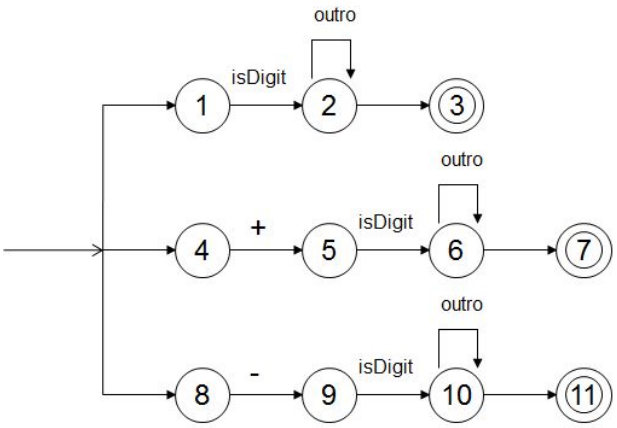
TP - Idealizar um autômato para reconhecer números inteiros;



```
FileStream infile;
        String number = "";

        infile = new System.IO.FileStream("Teste.txt",
                                  System.IO.FileMode.Open,
                                  System.IO.FileAccess.Read);
        int tam;
        char x;

        tam = (int)infile.Length;


        for (int i = 0; i < tam; i++)
        {
           x = (char)infile.ReadByte();
           if (char.IsDigit(x))
           {
              do
              {
                 number += x;
                 i++;
                 x = (char)infile.ReadByte();
              }
              while (char.IsDigit(x));
              Console.WriteLine("\n" + number);
              number = "";
           }
        }
        Console.ReadKey();
     }
```

Aula 4 - Autômatos Finitos II

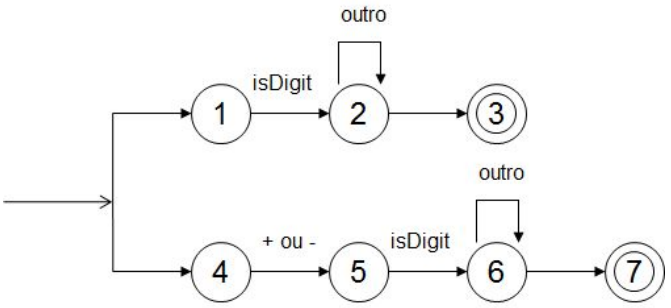
Idealizar um autômato que identifica números inteiros e seu respectivo sinal.




```
        FileStream infile;
          int tam,i;
          char x;
          String aux;

          infile = new
System.IO.FileStream("teste.txt",

System.IO.FileMode.Open,
 System.IO.FileAccess.Read);

          tam = (int)infile.Length;
          for (i = 0; i < tam; ++i)
          {
             aux = "";
             x = (char)infile.ReadByte();

             if (char.IsDigit(x))
             {
                while (char.IsDigit(x))
                {
                   aux = aux + x;
                   x = (char)infile.ReadByte();
                   ++i;
                }
                Console.WriteLine(aux);
             }
```

```
if (x == '+')
            {
               aux = aux + x;
               x = (char)infile.ReadByte();
               if (char.IsDigit(x))
               {
                  while (char.IsDigit(x))
                  {
                     aux = aux + x;
                     x = (char)infile.ReadByte();
                     ++i;
                  }
                  Console.WriteLine(aux);
               }
            }

            if (x == '-')
            {
               aux = aux + x;
               x = (char)infile.ReadByte();
               if (char.IsDigit(x))
               {
                  while (char.IsDigit(x))
                  {
                     aux = aux + x;
                     x = (char)infile.ReadByte();
                     ++i;
                  }
                  Console.WriteLine(aux);
```

Bullet Points: A redundância aqui está em colocar dois if para o sinal de - e +, o certo seria usar **x** == **-** || **x**== +.

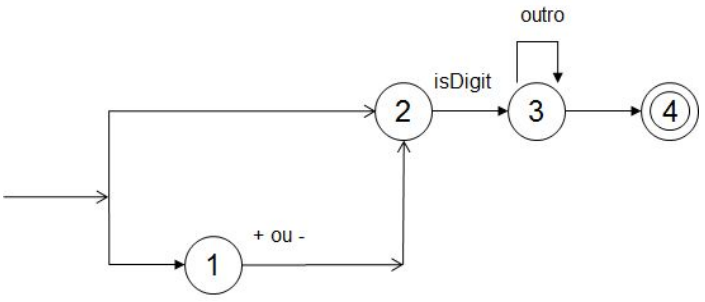



```
        FileStream infile;
        int tam;
        char x;
        String aux;

        infile = new
System.IO.FileStream("teste.txt",
                             System.IO.FileMode.Open,
                             System.IO.FileAccess.Read);

        tam = (int)infile.Length;
        for (int i = 0; i < tam; ++i)
        {
           aux = "";
           x = (char)infile.ReadByte();

           if (char.IsDigit(x))
           {
              while (char.IsDigit(x))
              {
                 aux = aux + x;
                 x = (char)infile.ReadByte();
                 ++i;
              }
              Console.WriteLine(aux);
           }

           if (x == '+' || x == '-')
           {
              aux = aux + x;
              x = (char)infile.ReadByte();
              if (char.IsDigit(x))
              {
```

```
            while (char.IsDigit(x))
            {
               aux = aux + x;
               x = (char)infile.ReadByte();
               ++i;
            }
            Console.WriteLine(aux);
         }
      }

   }
   infile.Close();
   Console.ReadKey();
```

Bullet Points : A redundância está em perguntar se é isDigit duas vezes.

```
FileStream infile;
   int tam;
   char x;
   String aux;

   infile = new
System.IO.FileStream("teste.txt",
                     System.IO.FileMode.Open,
                     System.IO.FileAccess.Read);

   tam = (int)infile.Length;
   for (int i = 0; i < tam; ++i)
   {
      aux = "";
      x = (char)infile.ReadByte();
```

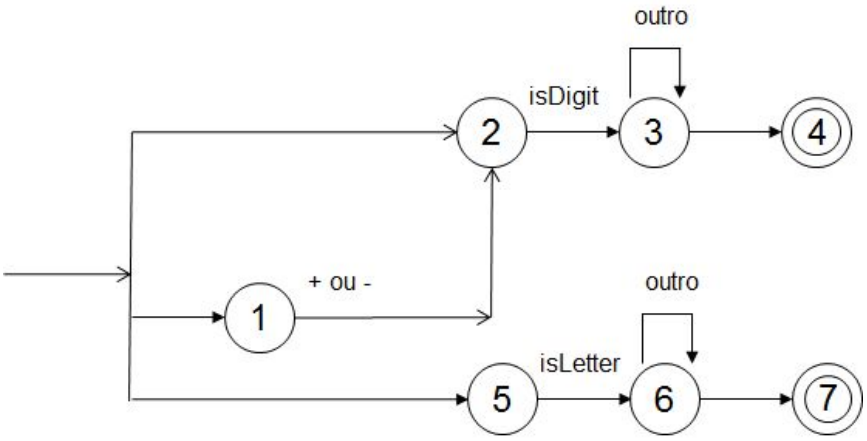


```
            if (x == '+' || x == '-')
            {
               aux = aux + x;
               x = (char)infile.ReadByte();
               ++i ---> acho q tá ok
            }

            if (char.IsDigit(x))
            {
               while (char.IsDigit(x))
               {
                  aux = aux + x;
                  x = (char)infile.ReadByte();
                  ++i;
               }
               Console.WriteLine(aux);
            }

         }
infile.Close();
Console.ReadKey();
```

TP3 - Encontrar String, número e identificá-los

```
            FileStream infile;
            int tam, linha;
            char x;
            String aux;

            infile = new System.IO.FileStream("teste.txt",
                                              System.IO.FileMode.Open,
                                              System.IO.FileAccess.Read);
            linha = 1;
            tam = (int)infile.Length;
            for (int i = 0; i < tam; ++i)
            {
```

```
                aux = "";
                x = (char)infile.ReadByte();

                if (char.IsLetter(x))
                {
                while (char.IsLetter(x))
                {
                      aux = aux + x;
                      x = (char)infile.ReadByte();
                       i++;
                }
                Console.WriteLine("Linha {0}: Palavra - {1} ", linha,
aux);

                }

                if (x == '+' || x == '-')
                {
                aux = aux + x;
                x = (char)infile.ReadByte();
                }

                if (char.IsDigit(x))
                {
                while(char.IsDigit(x))
                {
                      aux = aux + x;
                      x = (char)infile.ReadByte();
                      i++;
                }
                Console.WriteLine("Linha {0}: Número - {1}", linha,
aux);
                }

                  if (x == 10)
                {
                      linha++;
                }
          }
          infile.Close();
          Console.ReadKey();
```



Aula 5 - Autômatos Finitos III (+ e - dupla interpretação)

- Acrescentar a identificação de delimitadores na solução da semana passada.
- Ocorrerá um problema o sinal de + e - ora serão interpretados como delimitadores ora como parte integral de um número.

Poder do cérebro

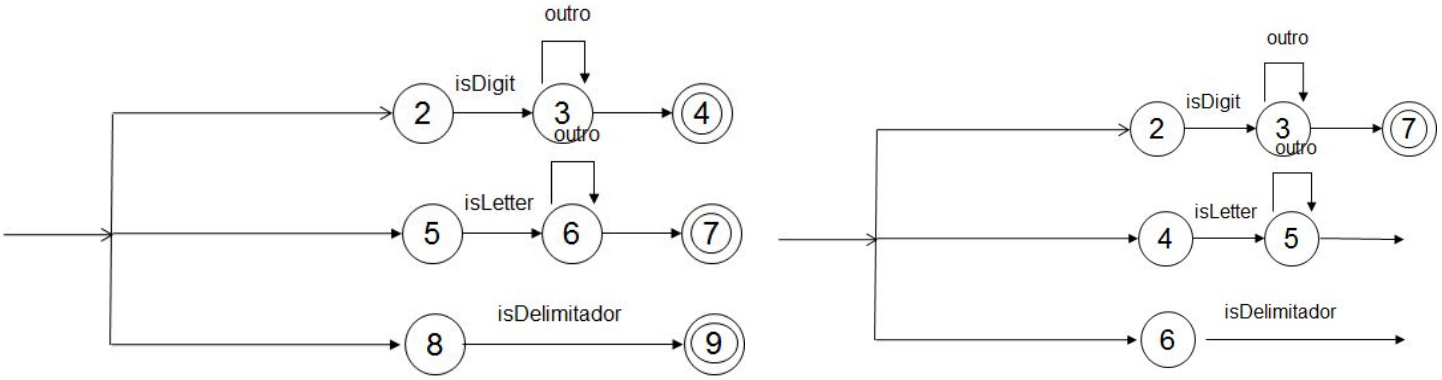
Os sinais de + e - devem ser interpretados como delimitadores ou como parte integral de um número?

Devem ser interpretados como delimitadores primeiramente (lexicamente, porque existem), depois sintaticamente, que verá se tem um número ao lado ou não, podendo definir se é parte integral ou apenas delimitador.

BULLET POINTS:

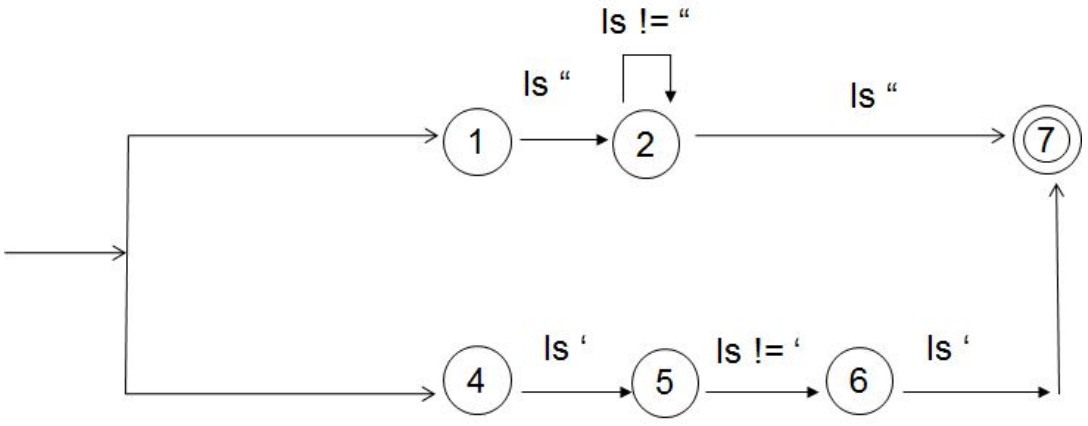
Em um arquivo texto, em que tenho **-+arquivo+-** ele **não reconhecerá o + como delimitador,** pois o que define o final de uma String é um **separador (outro caracter diferente de uma String)**.

Representação do autômato da aula passada identificando também delimitador (OBS: FALTANDO O CÓDIGO).



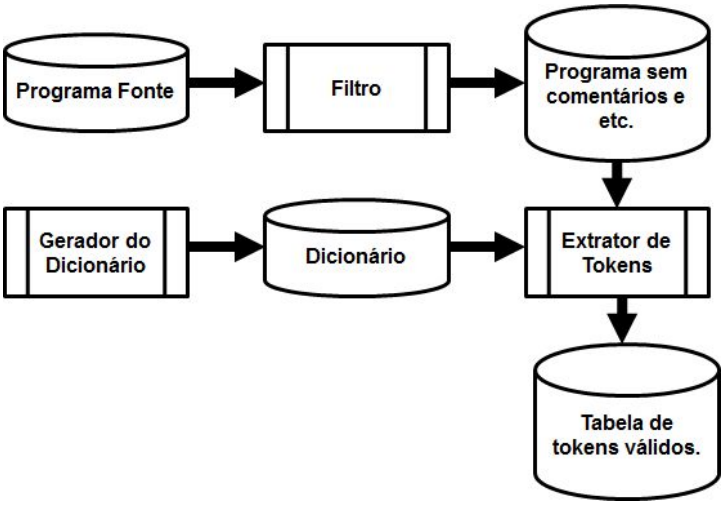
TP - 04 (Identificar caracter e string)

- *No C# um caracter deve aparecer entre aspas simples (apóstrofo) enquanto que uma string entre aspas duplas. Desenhe um autômato finito que seja capaz de identificar caracter e string dentro de um texto.*
- *Implementar tal autômato*



```
namespace TP4___StringChar
{
    class Program
    {
        static void Main(string[] args)
        {
              FileStream infile;

              int tam;
              char x;
              String aux = "";


              infile = new System.IO.FileStream("teste.txt", System.IO.FileMode.Open,
                                                System.IO.FileAccess.Read);

              tam = (int)infile.Length;

              for (int i = 0; i < tam; i++)
              {
              x = (char)infile.ReadByte();

              if (x == '"')
              {
                     i++;
                     x = (char)infile.ReadByte();
                     do
                     {
                     aux += x;
                     i++;
                     x = (char)infile.ReadByte();
                     }
                     while (x != '"');

                     Console.WriteLine("String : {0} ", aux);
                     aux = "";
              }

                     if (x == '\'')
                     {
                            i++;
                            x = (char)infile.ReadByte();
                            aux += x;
                            Console.WriteLine("Char : {0} ", aux);
                            aux = "";

                            i++;
                            x = (char)infile.ReadByte();
                     }

              }
                     Console.ReadKey();
              }
```

Aula 5 - Analisador Léxico



BULLET POINTS:

- O Extrator de token chamado mais formalmente de scanner localiza tokens inválidos e os retira do programa.
- *O filtro despreza comentários e identação.*
- O macrofluxo representa todo o desenho/representação do analisador léxico.

TP5 - Juntar extração de comentários/espaço

```
class Fonte
    {
        private static String pathnome;

        public static void setPathNome(String _pathnome) { pathnome = _pathnome; }
        public static String getPathNome() { return pathnome; }
       }
```

```
class FonteBLL
    {
        public static void Filtro()
        {
              FileStream infile, outfile;
              int tam;
              char x;

              infile = new System.IO.FileStream(Fonte.getPathNome(),
                                                System.IO.FileMode.Open,
                                                System.IO.FileAccess.Read);
```

```
            outfile = new System.IO.FileStream("PFTMP.txt",
                                               System.IO.FileMode.Create,
                                               System.IO.FileAccess.Write);
            tam = (int)infile.Length;
            for (int i = 0; i < tam; ++i)
            {
            x = (char)infile.ReadByte();
            if (x == '#')
            {
                   ++i;
                   do
                   {
                   x = (char)infile.ReadByte();
                   ++i;
                   }
                   while (x != '#');
            }
            else
                   if (x != ' ')
                   outfile.WriteByte((byte)char.ToUpper(x));
            }
            infile.Close();
            outfile.Close();
      }
     }
{
```

```
    class Erro
    {
        private static Boolean erro;
        private static String msg;

        public static void setErro(Boolean _erro)
        {
              erro = _erro;
        }

        public static void setErro(String _msg)
        {
              erro = true;
              msg = _msg;
        }

        public static Boolean getErro() { return erro; }
        public static String getMsg() { return msg; }
    }
}
```

```
FORM
private void arquivoToolStripMenuItem_Click(object sender, EventArgs e)
        {
              openFileDialog1.ShowDialog();
              this.Text = openFileDialog1.FileName; //this é o form
              análiseLéxicaToolStripMenuItem.Enabled = true;
        }

        private void análiseLéxicaToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Fonte.setPathNome(openFileDialog1.FileName); //pegar o caminho do arquivo

            FonteBLL.Filtro();

              if (Erro.getErro())
              {
              MessageBox.Show(Erro.getMsg());
              }
              else
              MessageBox.Show("Cópia Realizada com Sucesso!");
       }
```

Aula 7 - Inserção dos tokens no Banco de Dados (dicionário)

```
       ANALISE LÉXICA BLL
       private static void grava(int codigo, String tipo, int linha, String token)
        {
              TTokensValidos.setCodigo(codigo);
              TTokensValidos.setTipo(tipo);
              TTokensValidos.setLinha(linha);
              TTokensValidos.setTokensValidos(token);

              MeuCompiladorDAL.inseriUmTokenValido();
        }

        public static void Scanner()
        {
              FileStream infile;
              int tam;
              int k1 = 1;
              char x;
              String aux = "";

              infile = new System.IO.FileStream("pftmp.txt",
                                                System.IO.FileMode.Open,
                                                System.IO.FileAccess.Read);

              tam = (int)infile.Length;
              for (int i = 0; i < tam; ++i)
              {
              x = (char)infile.ReadByte();
```

```
            if (char.IsDigit(x))
            {
                  while (char.IsDigit(x))
                  {
                  aux = aux + x;
                  x = (char)infile.ReadByte();
                  ++i;
                  }
                  grava(1, "Inteiro", kl, aux);
                  //System.Windows.Forms.MessageBox.Show("Linha   "   +   kl   +   "
(Inteiro)..: " + aux);
                 aux = "";
            }

            if (char.IsLetter(x))
            {
                  aux = "";
                  while (char.IsLetter(x))
                  {
                  aux = aux + x;
                  x = (char)infile.ReadByte();
                  ++i;
                  }

                  grava(1, "String", kl, aux);
                  //System.Windows.Forms.MessageBox.Show("Linha   "   +   kl   +   "
(String)...: " + aux);
                  aux = "";
            }

            if (char.IsPunctuation(x) || char.IsSymbol(x))
                  grava(1, "Delimitador", kl, x.ToString());


            if (x == 13) ++kl;
            }
            infile.Close();
      }


      public static void filtro()
      {
            FileStream infile, outfile;
            int tam;
            char x;

            infile = new System.IO.FileStream(Fonte.getPathNome(),
                                          System.IO.FileMode.Open,
                                    System.IO.FileAccess.Read);

            outfile = new System.IO.FileStream("pftmp.txt",
                                          System.IO.FileMode.Create,
                                          System.IO.FileAccess.Write);
```

```
            tam = (int)infile.Length;
            for (int i = 0; i < tam; ++i)
            {
            x = (char)infile.ReadByte();
            if (x == '#')
            {
                  ++i;
                  do
                  {
                  x = (char)infile.ReadByte();
                  ++i;
                  }
                  while (x != '#');
            }
            else
                  if (x != ' ')
                  outfile.WriteByte((byte)char.ToUpper(x));
            }
            infile.Close();
            outfile.Close();
      }
```

```
class MeuCompiladorDAL
    {
          private static String strConexao = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=Compilador.mdb";
        private static OleDbConnection conn = new OleDbConnection(strConexao);
        private static OleDbCommand strSQL;
        private static OleDbDataReader result;

        public static void conecta()
        {
              try
              {
              conn.Open();
              }
              catch
              {
              Erro.setMsg("A conexão falhou!");
              return;
              }
        }

        public static void desconecta()
        {
              try
              {
              conn.Close();
              }
              catch
              {
              Erro.setMsg("Desconexão falhou!");
              return;
```

```
            }
        }

        public static void inseriUmTokenValido()
        {
            String sql = "insert into TTokensValidos (codigo, token, tipo, linha)
values ('" + TTokensValidos.getCodigo() + "','" + TTokensValidos.getTokensValidos() +
"','" + TTokensValidos.getTipo() + "', '" + TTokensValidos.getLinha() + "')";

            Erro.setErro(false);

            //Rodar comando SQL
            try
            {
            strSQL = new OleDbCommand(sql, conn);
            strSQL.ExecuteNonQuery();
            }
            catch
            {
            Erro.setMsg("Código duplicado, esse token já foi incluído!");
            }

        }
```

```
 namespace MeuCompilador
{
    class MeuCompiladorBLL
    {
        public static void compilarPrograma()
        {
            AnalisadorLexicoBLL.filtro();
        }

        public static void scanner()
        {
            AnalisadorLexicoBLL.scanner();
        }

        public static void conecta()
        {
            MeuCompiladorDAL.conecta();
        }

        public static void insereUmTokenValido()
        {
            MeuCompiladorDAL.insereUmTokenValido();
        }
        public static void desconecta()
        {
            MeuCompiladorDAL.desconecta();
        }}
```

```
   class TTokensValidos
    {
        private static int codigo;
        private static String tokensvalidos;
        private static String tipo;
        private static int linha;

        public static void setCodigo(int _codigo) { codigo = _codigo; }
        public static int getCodigo() { return codigo; }

         public static void setTokensValidos(String _tokensvalidos) { tokensvalidos =
_tokensvalidos; }
        public static String getTokensValidos() { return tokensvalidos; }

        public static void setTipo(String _tipo) { tipo = _tipo; }
        public static String getTipo() { return tipo; }

        public static void setLinha(int _linha) { linha = _linha; }
        public static int getLinha() { return linha; }
       }
```

```
class TTokens
    {
        private static int codigo;
        private static String tokens;

        public static void setCodigo(int _codigo) { codigo = _codigo; }
        public static int getCodigo() { return codigo; }

        public static void setTokens(String _tokens) { tokens = _tokens; }
        public static String getTokens() { return tokens; }

       }
```

```
private void compilarProgramaToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MeuCompiladorDAL.conecta();
              if (Erro.getErro())
              MessageBox.Show(Erro.getMsg());

              Fonte.setPathNome(openFileDialog1.FileName);

              if (Erro.getErro())
              {
              MessageBox.Show(Erro.getMsg());
              }
              else
              {
              FonteBLL.filtro();
              FonteBLL.Scanner();
              }
```

```
    }

    private void abrirProgramaToolStripMenuItem_Click(object sender, EventArgs e)
    {
          openFileDialog1.ShowDialog();
          this.Text = openFileDialog1.FileName;
        compilarProgramaToolStripMenuItem.Enabled = true;
    }

    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
          MeuCompiladorDAL.desconecta();
    }
```

EXBD

```
class LivroBLL
{
    public static void validaCodigo()
    {
        Erro.setErro(false);
        if (Livro.getCodigo().Equals(""))
        {
            Erro.setMsg("O código é de preenchimento obrigatório!");
            return;
        }
        LivroDAL.consultaUmLivro();
    }

    public static void validaDados()
    {
        Erro.setErro(false);
        if (Livro.getCodigo().Equals(""))
        {
            Erro.setMsg("O código é de preenchimento obrigatório!");
            return;
        }
        if (Livro.getTitulo().Equals(""))
        {
            Erro.setMsg("O título é de preenchimento obrigatório!");
            return;
        }
        if (Livro.getAutor().Equals(""))
        {
            Erro.setMsg("O autor é de preenchimento obrigatório!");
            return;
        }
        if (Livro.getEditora().Equals(""))
        {
            Erro.setMsg("A Editora é de preenchimento obrigatório!");
            return;
        }
```

```
            if (Livro.getAno().Equals(""))
            {
                Erro.setMsg("O ano é de preenchimento obrigatório!");
                return;
            }


            try
            {
                int.Parse(Livro.getAno());
            }
            catch (Exception)
            {
                Erro.setMsg("O valor do ano deve ser numérico!");
                return;
            }


            if (int.Parse(Livro.getAno()) <= 0)
            {
                Erro.setMsg("O valor do Ano deve ser numérico e positivo!");
                return;
            }
            LivroDAL.inseriUmLivro();

        }

    }
}
```

```
  class LivroDAL
    {
           private static String strConexao = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=BDLivros.mdb";
        private static OleDbConnection conn = new OleDbConnection(strConexao);
        private static OleDbCommand strSQL;
        private static OleDbDataReader result;

        public static void conecta()
        {
            try
            {
                conn.Open();
            }
            catch
            {
                Erro.setMsg("A conexão falhou!");
                return;
            }
        }

        public static void desconecta()
```

```
        {
            try
            {
                conn.Close();
            }
            catch
            {
                Erro.setMsg("Desconexão falhou!");
                return;
            }
        }

        public static void inseriUmLivro()
        {
             String aux = "insert into TabLivro(codigo,titulo,autor,editora,ano) values
('" + Livro.getCodigo() + "','" + Livro.getTitulo() + "','" + Livro.getAutor() + "','"
+ Livro.getEditora() + "','" + Livro.getAno() + "')";

            conecta();
            strSQL = new OleDbCommand(aux, conn);
            strSQL.ExecuteNonQuery();
            desconecta();
        }

        public static void consultaUmLivro()
        {
              String aux = "select * from TabLivro where codigo ='" + Livro.getCodigo()
+ "'";

            conecta();
            strSQL = new OleDbCommand(aux, conn);
            result = strSQL.ExecuteReader();
            Erro.setErro(false);
            if (result.Read())
            {
                Livro.setTitulo(result.GetString(1));
                Livro.setAutor(result.GetString(2));
                Livro.setEditora(result.GetString(3));
                Livro.setAno(result.GetString(4));
            }
            else
                Erro.setMsg("Livro não cadastrado.");
            desconecta();
        }


    }
}
```

```
    private void button2_Click(object sender, EventArgs e) //Limpar
    {
        textBox1.Text = "";
        textBox2.Text = "";
        textBox3.Text = "";
        textBox4.Text = "";
        textBox5.Text = "";
    }

    private void button1_Click(object sender, EventArgs e) //Salvar
    {
        Livro.setCodigo(textBox1.Text);
        Livro.setTitulo(textBox2.Text);
        Livro.setAutor(textBox3.Text);
        Livro.setEditora(textBox4.Text);
        Livro.setAno(textBox5.Text);

        LivroBLL.validaDados();

        if (Erro.getErro())
            MessageBox.Show(Erro.getMsg());
        else
            MessageBox.Show("Dados inseridos com sucesso!");
    }

    private void button3_Click(object sender, EventArgs e) //Ler
    {
        Livro.setCodigo(textBox1.Text);
        LivroBLL.validaCodigo();
        if (Erro.getErro())
            MessageBox.Show(Erro.getMsg());
        else
        {
            textBox1.Text = Livro.getCodigo();
            textBox2.Text = Livro.getTitulo();
            textBox3.Text = Livro.getAutor();
            textBox4.Text = Livro.getEditora();
            textBox5.Text = Livro.getAno();
        }
    }
}
```

Bullet Points: Eu posso passar do valor da instrução usando do while.

Aula 09

```
namespace MeuCompilador
{
    class Token
    {
        private static String codigo;
        private static String token;
        private static String tipo;
        private static String linha;

        public static void setCodigo(String _codigo) { codigo = _codigo; }
        public static void setToken(String _token) { token = _token; }
        public static void setTipo(String _tipo) { tipo = _tipo; }
        public static void setLinha(String _linha) { linha = _linha; }
        public static String getCodigo() { return codigo; }
        public static String getToken() { return token; }
        public static String getTipo() { return tipo; }
        public static String getLinha() { return linha; }
    }
}
namespace MeuCompilador
{
    class MeuCompiladorDAL
    {
            private static String strConexao = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=Compilador.mdb";
        private static OleDbConnection conn = new OleDbConnection(strConexao);
        private static OleDbCommand strSQL;
        private static OleDbDataReader result;

        public static void conecta()
        {
              Erro.setErro(false);
              try
              {
              conn.Open();
              }
              catch
              {
              Erro.setErro("A conexão falhou!");
              return;
              }
        }

        public static void desconecta()
        {
              conn.Close();
        }

        public static void inseriUmTokenValido()
        {
```

```
            String aux = "insert into TTokensValidos(codigo,token,tipo,linha) values
(" + Token.getCodigo() + ",'" + Token.getToken() + "','" + Token.getTipo() + "'," +
Token.getLinha() + ")";

            strSQL = new OleDbCommand(aux, conn);
            strSQL.ExecuteNonQuery();
        }
        public static void consultaUmToken()
        {
            OleDbDataReader result;
            String aux = "select * from TTokens where Token ='" + Token.getToken() +
"'";

            strSQL = new OleDbCommand(aux, conn);
            result = strSQL.ExecuteReader();
            Erro.setErro(false);
            if (result.Read())
            Token.setCodigo("" + result.GetInt32(0));
            else
            Erro.setErro("Linha " + Token.getLinha() + ": " + Token.getToken() + "
(Token não identificado)");
        }
        public static void deletaTTokensValidos()
        {
            String aux = "delete * from TTokensValidos";

            strSQL = new OleDbCommand(aux, conn);
            strSQL.ExecuteNonQuery();
        }

      }
   class AnalisadorLexicoBLL
    {
        public static void filtro()
        {
            FileStream infile, outfile;
            int tam;
            char x;

            infile = new System.IO.FileStream(ProgramaFonte.getPathNome(),
                                              System.IO.FileMode.Open,
                                              System.IO.FileAccess.Read);
            outfile = new System.IO.FileStream("pftmp.txt",
                                               System.IO.FileMode.Create,
                                              System.IO.FileAccess.Write);

            tam = (int)infile.Length;
            for (int i = 0; i < tam; ++i)
            {
            x = (char)infile.ReadByte();
               if (x == '#')
            {
                   ++i;
                   do
```

```
            {
            x = (char)infile.ReadByte();
            ++i;
            }
            while (x != '#');
        }
      else
            if (x != ' ')
            outfile.WriteByte((byte)char.ToUpper(x));
      }
      infile.Close();
      outfile.Close();
}

public static void scanner()
{
      FileStream infile;
      int tam;
      int kl = 1;
      char x;
      String aux="";

      infile = new System.IO.FileStream("pftmp.txt",
                                        System.IO.FileMode.Open,
                                        System.IO.FileAccess.Read);

      MeuCompiladorDAL.deletaTTokensValidos();
      tam = (int)infile.Length;
      for (int i = 0; i < tam; ++i)
      {
      x = (char)infile.ReadByte();

      if (char.IsDigit(x))
      {
            while (char.IsDigit(x))
            {
            aux = aux + x;
            x = (char)infile.ReadByte();
            ++i;
            }
            Token.setCodigo("200");
            Token.setToken(aux);
            Token.setTipo("Inteiro");
            Token.setLinha(kl.ToString());
            MeuCompiladorDAL.inseriUmTokenValido();
         aux = "";

      }

      if (char.IsLetter(x))
      {
            aux = "";
            while (char.IsLetter(x))
            {
```

```
                aux = aux + x;
                x = (char)infile.ReadByte();
                ++i;
                 }
                Token.setToken(aux);
                Token.setTipo("String");
                Token.setLinha(kl.ToString());
                MeuCompiladorDAL.consultaUmToken();
                if (Erro.getErro())
                return;
                else
                MeuCompiladorDAL.inseriUmTokenValido();
                aux = "";
            }
            if (char.IsPunctuation(x) || char.IsSymbol(x))
            {
                Token.setToken(x.ToString());
                Token.setTipo("Delimitador");
                Token.setLinha(kl.ToString());
                MeuCompiladorDAL.consultaUmToken();
                if (Erro.getErro())
                   return;
                else
                MeuCompiladorDAL.inseriUmTokenValido();
            }

            if (x == 13) ++kl;

            }
            infile.Close();
}
```

TP

Do jeito que está os tokens encontrados independentemente de serem válidos ou não sempre serão inseridos na tabela de “Tokens Validos”. No entanto somente os tokens válidos é que deveriam sê-lo.

Assim volte a versão III e só insira o token na tabela TTokensValidos se este for de fato um token valido. Para tanto antes da gravação você deverá consultar a tabela TTokens a fim de verificar sua existência.

Perceba também que em nossa solução os tokens são gravados sempre com o código 200, agora estes deverão ser gravados com seus códigos de fato, que serão obtidos durante a consulta a tabela TTokens, o nosso dicionário.

Aula 10 - Análise Sintática

- É a etapa posterior a análise Sintática;
- Não há mais erros léxicos (tokens inválidos);
- Seu objetivo é verificar tokens fora da sequência;

TP

Teremos que ler sequencialmente a tabela TTokensValidos a fim de validar ou não as sequências.

Assim vamos começar fazendo esta leitura sequencial.

- public static void populaDR();
- public static void leUmTokenValido();

```
        class MeuCompiladorDAL
    {
       public static void populaDR()
        {
              String aux = "select * from TTokensValidos";

              strSQL = new OleDbCommand(aux, conn);
              resultAS = strSQL.ExecuteReader();
        }

        public static void leUmTokenValido()
        {
              Erro.setErro(false);
              if (resultAS.Read())
              {
              Token.setCodigo("" + resultAS.GetInt32(0));
              Token.setToken(resultAS.GetString(1));
              Token.setTipo(resultAS.GetString(2));
              Token.setLinha("" + resultAS.GetInt32(3));
              }
              else
              Erro.setErro(true);

       }
}
  class AnalisadorSintaticoBLL
    {
        public static void testeAnaliseSintatica()
        {
              MeuCompiladorDAL.populaDR();

              MeuCompiladorDAL.leUmTokenValido();
              while (Erro.getErro() == false)
              {
              System.Windows.Forms.MessageBox.Show(Token.getToken());
              MeuCompiladorDAL.leUmTokenValido();
              }
              Erro.setErro(false);
        }
    }
}
    class MeuCompiladorBLL
    {
        public static void compilarPrograma()
        {
              // Procedimento para Análise Léxica do meu código
```

```
            AnalisadorLexicoBLL.filtro();
            AnalisadorLexicoBLL.scanner();
            if (Erro.getErro()) return;

            // Procedimento para Análise Sintática do meu código

            AnalisadorSintaticoBLL.testeAnaliseSintatica();
        }
```

Aula 11 - Gabarito Sintático com vetor

```
namespace MeuCompilador
{
        class AnalisadorSintaticoBLL
        {
        private static int[] lt = { 0, 100, 101, 103 };
        private static int[] pc = { 1, 100, 200, 102, 200, 101, 103 };
        private static int[] ed = { 2, 100, 200, 101, 103 };
        private static int[] f = { 3, 100, 101, 103 };

        public static void validaSequencia(int[] _seq)
        {
                Erro.setErro(false);
                for (int i = 1; i < _seq.Length; ++i)
                {
                MeuCompiladorDAL.leUmTokenValido();
                if (_seq[i] != Int32.Parse(Token.getCodigo()))
                {
                        Erro.setErro("Linha " + Token.getLinha() + "- token inesperado: "
+ Token.getToken());
                        return;
                }
            }
        }

        public static void analiseSintatica()
        {

                MeuCompiladorDAL.populaDR(); //select

                MeuCompiladorDAL.leUmTokenValido(); //seta os dados da tabela na classe
                while (Erro.getErro() == false)
                {
                switch (Int32.Parse(Token.getCodigo())) //recebe o codigo
                {
                        case 0: validaSequencia(lt); if (Erro.getErro()) return; break;
                        case 1: validaSequencia(pc); if (Erro.getErro()) return; break;
                        case 2: validaSequencia(ed); if (Erro.getErro()) return; break;
                        case 3: validaSequencia(f); if (Erro.getErro()) return; break;
                        default: Erro.setErro("Linha " + Token.getLinha() + "- Comando
esperado: " + Token.getToken()); return;
                }
```

```
                MeuCompiladorDAL.leUmTokenValido();
                }
                Erro.setErro(false);
        }

        }
}
```

Solução com Array Bidimensional (Matriz)

```
{
        class AnalisadorSintaticoBLL
        {
        private static int[][] gabarito = new int[][] {new int[] { 0, 100, 101, 103 },
                                        new int[] { 1, 100, 200, 102, 200, 101, 103 },
                                        new int[] { 2, 100, 200, 101, 103 },
                                        new int[] { 3, 100, 101, 103 }}; //matriz

        public static void validaSequencia(int[] _seq)
        {
                Erro.setErro(false);
                for (int i = 1; i < _seq.Length; ++i)
                {
                MeuCompiladorDAL.leUmTokenValido();
                if (_seq[i] != Int32.Parse(Token.getCodigo()))
                {
                        Erro.setErro("Linha " + Token.getLinha() + "- token inesperado: "
+ Token.getToken());
                        return;
                }
                }
        }

        public static void analiseSintatica()
        {

                MeuCompiladorDAL.populaDR();

                MeuCompiladorDAL.leUmTokenValido();
                while (Erro.getErro() == false)
                {
                validaSequencia(gabarito[int.Parse(Token.getCodigo())]);
                if (Erro.getErro()) return;
                MeuCompiladorDAL.leUmTokenValido();
                }
           Erro.setErro(false);
        }

        //Com a matriz não preciso do switch porque ela lê a linha e a coluna.
        }
}
```

```
Aula 12
namespace MeuCompilador
{
	class AnalisadorSintaticoBLL
	{
	public static void validaSequencia()
	{
		Erro.setErro(false);
		MeuCompiladorDAL.populaGabarito(int.Parse(Token.getCodigo()));
		MeuCompiladorDAL.leGabarito();
		if (Erro.getErro())
		{
		Erro.setErro("Linha " + Token.getLinha() + "- token inesperado: " +
Token.getToken());
		return;
		}

		while (!Erro.getErro())
		{
		if (Token.getCodigo() != Gabarito.getInfo())
		{
			Erro.setErro("Linha " + Token.getLinha() + "- token inesperado: "
+ Token.getToken());
			return;
		}
		if (Gabarito.getNext() != "eof") MeuCompiladorDAL.leUmTokenValido();
		MeuCompiladorDAL.leGabarito();
		}
		Erro.setErro(false);
	}

	public static void analiseSintatica()
	{

		MeuCompiladorDAL.populaDR();

		MeuCompiladorDAL.leUmTokenValido();
		while (Erro.getErro() == false)
		{
		validaSequencia();
		if (Erro.getErro()) return;
		MeuCompiladorDAL.leUmTokenValido();
		}
		Erro.setErro(false);
	}

	}
}
```

code	prior	info	next
0	bof	0	100
0	0	100	101
0	100	101	103
0	101	103	eof
1	bof	1	100
1	1	100	200
1	100	200	102
1	200	102	200
1	102	200	101
1	200	101	103
1	101	103	eof
2	bof	2	100
2	2	100	200
2	100	200	101
2	200	101	103
2	101	103	eof
3	bof	3	100
3	3	100	101
3	100	101	103
3	101	103	eof

Aula 13 - Análise Semântica

- **Verifica os erros semânticos, dados e/ou valores que não façam sentido no código fonte**, além de **preparar a próxima fase da compilação que é a tradução** do código fonte para código objeto (no nosso caso o executável).

Exemplos:

- **Uma operação aritmética envolvendo tipos de dados diferentes**;
- **Atribuição de um literal para outro tipo**, como **um inteiro em uma string** ou vice-versa.

TP1 - Deleta delimitadores

MeuCompiladorDAL

```
public static void deletaDelimitadores()
{
    String aux = "delete from TTokensValidos where Tipo = 'Delimitador'";
    strSQL = new OleDbCommand(aux,conn);
    strSQL.ExecuteNonQuery();
}
```

AnalisadorSemanticoBLL

```
class AnalisadorSemanticoBLL
{
    public static void analiseSemantica()
    {
        MeuCompiladorDAL.deletaDelimitadores();
    }
}
```

MeuCompiladorBLL

```
class MeuCompiladorBLL
{
    public static void compilarPrograma()
    {
        // Procedimento para Análise Léxica do meu código

        AnalisadorLexicoBLL.filtro();
        AnalisadorLexicoBLL.scanner();
        if (Erro.getErro()) return;

        // Procedimento para Análise Sintática do meu código
```

```
            AnalisadorSintaticoBLL.analiseSintatica();

              // Procedimento para Análise Semântica do meu código
              AnalisadorSemanticoBLL.analiseSemantica();
        }
      }
```

Aula 14 - Análise Semântica II

Vamos a classe AnalisadorSemanticoBLL, perceba que nesta classe temos o método: **analiseSemantica()** que apenas **elimina os delimitadores** de nossa tabela.

Complemente esse método que deverá **validar os valores associados aos comandos POSICIONACURSOR e ESCREVEDIGITO** de acordo com as seguintes regras:

No caso do POSICIONACURSOR deveremos analisar o primeiro inteiro (coluna) e verificar se ele esta entre 1 e 80; depois analisar o segundo inteiro (linha) e verificar se ele esta entre 1 e 24;

No caso do ESCREVEDIGITO o inteiro associado deverá ser verificado se esta entre 0 e 9.

```
public static void analiseSemantica()
        {
              Erro.setErro(false);
              MeuCompiladorDAL.deletaDelimitadores();

              MeuCompiladorDAL.populaDR();
              MeuCompiladorDAL.leUmTokenValido();
              while (!Erro.getErro())
              {
              if (int.Parse(Token.getCodigo()) == 1)
              {
                    MeuCompiladorDAL.leUmTokenValido();
                    validacao(1, 80);
                    if (Erro.getErro()) return;
                    MeuCompiladorDAL.leUmTokenValido();
                    validacao(1, 24);
                    if (Erro.getErro()) return;
              }
              if (int.Parse(Token.getCodigo()) == 2)
              {
                    MeuCompiladorDAL.leUmTokenValido();
                    validacao(0, 9);
                    if (Erro.getErro()) return;
              }
              MeuCompiladorDAL.leUmTokenValido();
              }
        }
        private static void validacao(int contadorinicial, int contadorfinal)
```

```
    {
            if      (int.Parse(Token.getToken())      <      contadorinicial      ||
int.Parse(Token.getToken()) > contadorfinal)
            Erro.setErro("Valor fora da faixa");
       }

            public static void populaDR()
        {
            String aux = "select * from TTokensValidos";

            strSQL = new OleDbCommand(aux, conn);
            resultAS = strSQL.ExecuteReader();
        }

     public static void leUmTokenValido()
       {
            Erro.setErro(false);
            if (resultAS.Read())
            {
            Token.setCodigo("" + resultAS.GetInt32(0));
            Token.setToken(resultAS.GetString(1));
            Token.setTipo(resultAS.GetString(2));
            Token.setLinha("" + resultAS.GetInt32(3));
            }
            else
            Erro.setErro(true);
       }
```

Aula 15 - Análise Semântica III

- A solução proposta não é a ideal pois nela encontramos um switch-case que questiona por todos os “comandos” que recebem argumentos;
- Além disso, em todos os casos executamos o mesmo procedimento;
- E o pior se o “comando” em questão possuir mais de um argumento repetiremos o procedimento tantas vezes quanto for o número de argumentos.

Nova Solução

Analisador Semântico BLL

```
class AnalisadorSemanticoBLL
    {
        public static void analiseSemantica()
        {
            Erro.setErro(false);
            MeuCompiladorDAL.deletaDelimitadores();
            MeuCompiladorDAL.populaDR();
```

```
                MeuCompiladorDAL.leUmTokenValido();
                while (Erro.getErro() == false)
                {
                int aux = MeuCompiladorDAL.leQtdArgumentos();
                for (int i = 0; i < aux; ++i)
                {
                        ArgLim.setCodigo(Token.getCodigo());
                        ArgLim.setposicao("" + i);
                        MeuCompiladorDAL.leUmLimite();
                        MeuCompiladorDAL.leUmTokenValido();
                        if  (int.Parse(Token.getToken())  <  int.Parse(ArgLim.getminimo())
|| int.Parse(Token.getToken()) > int.Parse(ArgLim.getmaximo()))
                        {
                        Erro.setErro("Linha " + Token.getLinha() + ": valor fora da faixa
(" + Token.getToken() + ")");
                        return;
                        }
                  }
                MeuCompiladorDAL.leUmTokenValido();
                }
                Erro.setErro(false);
         }
        }
```

Class ArgLimites

```
    class ArgLim
    {
        private static String codigo;
        private static String posicao;
        private static String minimo;
        private static String maximo;

        public static void setCodigo(String _codigo) { codigo = _codigo; }
        public static void setposicao(String _posicao) { posicao = _posicao; }
        public static void setminimo(String _minimo) { minimo = _minimo; }
        public static void setmaximo(String _maximo) { maximo = _maximo; }
        public static String getCodigo() { return codigo; }
        public static String getposicao() { return posicao; }
        public static String getminimo() { return minimo; }
        public static String getmaximo() { return maximo; }
       }
```

MeuCompiladorDAL

```
       public static int leQtdArgumentos()
        {
              String  aux  =  "select  *  from  TQtdArgumentos  where  Codigo  ="  +
Token.getCodigo();

              strSQL = new OleDbCommand(aux, conn);
              resultqtdarg = strSQL.ExecuteReader();
              resultqtdarg.Read();
              return resultqtdarg.GetInt32(1);
        }
```

```
        public static void leUmLimite()
        {
              String   aux   =   "select   *   from   TArgLimites   where   codigo   ="   +
ArgLim.getCodigo() + " and posicao =" + ArgLim.getposicao();

              strSQL = new OleDbCommand(aux, conn);
              resultlimites = strSQL.ExecuteReader();
              if (resultlimites.Read())
              {
                ArgLim.setminimo("" + resultlimites.GetInt32(2));
              ArgLim.setmaximo("" + resultlimites.GetInt32(3));
              }
       }
```

```
   class AnalisadorSemanticoBLL
    {
        public static void analiseSemantica()
        {
              Erro.setErro(false);
              MeuCompiladorDAL.deletaDelimitadores();
              MeuCompiladorDAL.populaDR();

              MeuCompiladorDAL.leUmTokenValido();
              while (Erro.getErro() == false)
              {
              switch (Token.getCodigo())
                {
                    case "1":
                    MeuCompiladorDAL.leUmTokenValido();
                    if        (int.Parse(Token.getToken())        <        1        ||
int.Parse(Token.getToken()) > 80)
                    {
                          Erro.setErro("Linha  "  +  Token.getLinha()  +  "-  valor  fora
da faixa: " + Token.getToken()); return;
                    }
                    MeuCompiladorDAL.leUmTokenValido();
                    if        (int.Parse(Token.getToken())        <        1        ||
int.Parse(Token.getToken()) > 24)
                    {
                          Erro.setErro("Linha  "  +  Token.getLinha()  +  "-  valor  fora
da faixa: " + Token.getToken()); return;
                    }
                    break;
                    case "2":
                    MeuCompiladorDAL.leUmTokenValido();
                    if        (int.Parse(Token.getToken())        <        0        ||
int.Parse(Token.getToken()) > 9)
                    {
                          Erro.setErro("Linha  "  +  Token.getLinha()  +  "-  valor  fora
da faixa: " + Token.getToken()); return;
                    }
                    break;
```

```
            }
            MeuCompiladorDAL.leUmTokenValido();
            }
            Erro.setErro(false);
        }
    }
```

Aula 16 - Asssembly

No processador variáveis são registradores.

Não existe if no mundo dos processadores, somente a pergunta “Deu zero?”

ax = acumulador	cx = contador	dx = dado

Para evitar o desperdício, os registradores são divididos ao meio. Por exemplo, para gravar uma letra é necessário 8 bits e cada registrador contém 16 bits, então o registrador é dividido ao meio pela metade alta (HIGH) e pela metade baixa (LOW), cada uma com 8 bits.

	HIGH	**LOW**
AX	**AH**	**AL**
BX	**BH**	**BL**
CX	**CH**	**CL**
DX	**DH**	**DL**
	16 BITS	

Segmento - endereço.
Offset - O deslocamento é utilizado para determinar uma posição dentro do bloco do segmento.

O segmento poderia ser comparado a um vetor e o offset ao índice ou posição.

TP 4 - ASSEMBLY

1. Em Assembly faça um programa capaz de posicionar o cursor na linha 10, coluna 40 do nosso monitor e então lá exibir a letra “S”.

```
MOV AH,02 //função para definir a posição do cursor;
MOV BH,00 //define a página
MOV DH,10 //define a linha
MOV DL,40 //define a coluna
INT 10h //sempre faço uma interrupção
MOV AH,09 //função que escreve caracter e atributo na posição do cursor
MOV AL,'S' //caractere que desejo escrever
MOV BH,00 //define a página
MOV BL,02 //define o texto ou cor;
MOV CX,01 //define o número de vezes que o caracter será escrito
INT 10h
```

2. Em Assembly faça um programa capaz de colocar a tela em modo gráfico e então na linha 20 traçar uma reta da coluna 20 até a coluna 100.

```
MOV AH,0 //função para definir o modo gráfico
MOV AL, 13h //função para definir a resolução do monitor
INT 10h
MOV CX,20 //define a coluna já no valor 20
VOLTA:
MOV AH,0Ch //escreve pixel no modo gráfico
MOV AL,03 //cor do pixel (nesse caso é verde)
MOV BH,00 //define a página
MOV DX,20 //define a coluna
INT 10h
INC CX //incremento o contador
CMP CX,101 //comparo o valor até dar 0
JNZ VOLTA
```

Exercícios de Revisão

1. Faça um progama capaz de atribuir 30 caracteres <space>, código ASCII 32 ou 20_{16}, para a partir do endereço 7000_{16}:0000_{16}.

```
153C:0100  BB0070          MOV  BX,7000
153C:0103  8EDB            MOV  DS,BX
153C:0105  BB0000          MOV  BX,0000
153C:0108  B91E00          MOV  CX,001E
153C:010B  B220            MOV  DL,20
153C:010D  8817            MOV  [BX],DL
153C:010F  43         INC  BX
153C:0110  49         DEC  CX
153C:0111  75FA            JNZ  010D
153C:0113  CD20            INT  20
```

2. (Esse caiu em prova) A partir do endereço 700016:010016 temos uma sequência de 10 bytes armazenados. Crie um programa que seja capaz de contar quantos “A” existem entre esses bytes, ao final devemos ter esse número dentro do registrador AX. Lembre-se que o código ASCII da letra “A” é 65 ou 4116.

```
1588:0100  BB0070          MOV  BX,7000
1588:0103  8EDB            MOV  DS,BX
1588:0105  BB0001          MOV  BX,0100
1588:0108  B90A00          MOV  CX,000A
1588:010B  B80000          MOV  AX,0000
1588:010E  8A17            MOV  DL,[BX]
1588:0110  80FA41          CMP  DL,41
1588:0113  7406            JZ   011B
1588:0115  43         INC  BX
1588:0116  49              DEC  CX
1588:0117  75F5            JNZ  010E
1588:0119  CD20            INT  20
1588:011B  40         INC  AX
1588:011C  EBF7            JMP  0115
```

Aula 17 - Assembly

1. Em Assembly faça um programa capaz de colocar a tela em modo gráfico e então desenhar um retângulo entre as seguintes coordenadas: (20,20) e (100,100).

```
MOV AH,00 // função para colocar em modo vídeo;
MOV AL,13h // Quanto de vídeo tenho de colocar;
INT 10h // interrupção da bios

MOV CX,20 // move a coluna para a posição 20
MOV AH,0Ch // função para escrever em modo de vídeo
MOV AL,03 // cor do pixel
MOV BH,00 // número da página

loop1:    // vou construindo as duas linhas
MOV DX,20 //move a linha para a posição 20
INT 10h // interrompe a bios

MOV DX,100 //move a linha para a posição 100
INT 10h //interrompe a bios

INC CX // incremento a coluna
CMP CX,101 //comparo a coluna com 101
JNZ loop1

MOV DX,20 // movo a linha para o 20
loop2:
MOV CX,20 //movo a coluna para o 20
INT 10h //interrompo a bios

MOV CX,100 //movo a coluna para o 100
INT 10h //interrompo a bios

INC DX //incremento a linha
CMP DX,101 //comparo a linha com 101        JNZ loop2                    RET
```

2. Em Assembly faça um programa capaz de receber via teclado 6 caracteres, estes caracteres deverão ecoar no vídeo.

```
MOV AH,00 ; colocar no modo vídeo
MOV AL,13h ; definir o modo do monitor
INT 10h ; interromper a bios

MOV CX,0 ; inicio o contador em 6
loop1:
INC CX
MOV AH,00 ;coloco no modo vídeo
INT 16h    ; para capturar eventos do teclado
MOV AH,0Eh  ; movo para função escrever por teletipo
MOV BH,00 ; defino a página 0
INT 10h   ; interromper bios
CMP CX,6    ;comparo com o 6
JNZ loop1:
ret
```

3. Idem ao exercício anterior porém a entrada deverá ser concluída quando você pressionar <Enter>.

```
MOV AH,00 ; colocar no modo video
MOV AL,13h ; definir o modo do monitor
INT 10h ; interromper a bios
loop1:
MOV AH,00 ;coloco no modo video
INT 16h    ; para capturar eventos do teclado
MOV AH,0Eh  ; movo para funcao escrever por teletipo
MOV BH,00 ; defino a pagina 0
INT 10h   ; interromper bios
CMP AL,13    ;comparo com o enter
JNZ loop1:
ret
```

Aula 18 - Asssembly

1. Voltando ao Console.ReadLine() do C# perceba que ele nunca aparece isolado em um código seu retorno é sempre atribuído a uma variável, por exemplo, *nome = Console.ReadLine();*. Apesar da nossa rotina emular o funcionamento do método ReadLine ela não armazena os caracteres digitados, que os são apenas exibidos em vídeo. Assim volte a rotina e faça com que a mesma armazene os dados digitados em uma variável localizada no endereço 5000h:0000h. Lembre-se que como ponteiro de memória você deve usar o par DS:BX.

```
MOV BX,5000
MOV DS,BX
MOV BX,0000h

MOV AH,00
MOV AL,13h
INT 10h

loop1:
MOV AH,00
INT 16h
MOV AH,0Eh
MOV BH,00
INT 10h
CMP AL,13
JZ fim:

MOV [BX], AL
INC BX

PUSH BX
MOV AH,0Eh
MOV BH,00
INT 10h
POP BX
JMP loop1
fim:
ret
```

Interrupção

1. INT 0x10 is used for screen manipulation
    - AH=0x00 -> set video mode
    - AX=0x1003 -> Set Blinking mode
    - AH=0x13 -> write string
    - AH=0x03 -> get cursor position
2. INT 0x13 is for storage (HDD and FDD)
    - AH=0x42 -> DISK READ
    - AH=0x43 -> DISK WRITE
3. INT 0x16 is for Keyboard control and read:
    - AH=0x00 -> GetKey
    - AH=0x03 -> Set typematic rate and delay

SP (STACK POINTER) - pilha do processador
Push - insere na pilha do processador
Pop - retira da pilha do processador e coloca no registrador

BACKSPACE

```
mov bx, 5000h
mov ds,bx
mov bx, 0000h
mov dh, 01  ;linha
mov dl, 00  ;coluna

loop:

mov ah, 00  ;le caracter d teclado
int 16h

cmp al, 13
jz fim

cmp al, 08
jz apaga

mov [bx], al
inc bx

call mostra
inc dl

jmp loop

apaga:
dec bx
mov [bx],00
dec dl

mov al,32
call mostra
jmp loop

fim:
ret

mostra:
push bx
mov ah, 02h ;para definir posicao do cursor
mov bh, 00
int 10h

mov ah, 0ah ;escreve caracter na posicao do cursor
mov bh, 00
mov cx, 1
int 10h
pop bx
ret
```

Aula 19 - Gerando Executável

```
    class GeradorExecutavelBLL
       {
       public static void geraExecutavel()
       {
              Erro.setErro(false);
              MeuCompiladorDAL.populaDR();
              MeuCompiladorDAL.leUmTokenValido();

              while (!Erro.getErro())
              {
              if (Token.getToken() == "LIMPATELA")
criaArquivo("C:\\Users\\RAFA\\Downloads\\Bim01Semana07\\LT.COM");
              else
                     if (Token.getToken() == "POSICIONACURSOR")
criaArquivo("C:\\Users\\RAFA\\Downloads\\Bim01Semana07\\PC.COM");
                     else
                     if (Token.getToken() == "ESCREVEDIGITO")
criaArquivo("C:\\Users\\RAFA\\Downloads\\Bim01Semana07\\ED.COM");
                     else if (Token.getToken() == "FIM")
criaArquivo("C:\\Users\\RAFA\\Downloads\\Bim01Semana07\\F.COM");

              MeuCompiladorDAL.leUmTokenValido();
              }

       }

       private static void criaArquivo(String filename) {

              FileStream infile, outfile;
              int tam;
              char x;

              infile = new System.IO.FileStream(filename,
                                                System.IO.FileMode.Open,
                                                System.IO.FileAccess.Read);

              outfile = new System.IO.FileStream("programa.com",
                                   System.IO.FileMode.Append,
                                   System.IO.FileAccess.Write);

              tam = (int)infile.Length;

              for (int i = 0; i < tam; i++)
              {
              x = (char)infile.ReadByte();
              outfile.WriteByte((byte)(x));
              }

              infile.Close(); outfile.Close();}}
```

```
class AnalisadorLexicoBLL
{
    public static void filtro()
    {
        FileStream infile, outfile;
        int tam;
        char x;

        infile = new System.IO.FileStream(ProgramaFonte.getPathNome(),
                                          System.IO.FileMode.Open,
                                          System.IO.FileAccess.Read);
        outfile = new System.IO.FileStream("pftmp.txt",
                                          System.IO.FileMode.Create,
                                          System.IO.FileAccess.Write);

        tam = (int)infile.Length;
        for (int i = 0; i < tam; ++i)
        {
            x = (char)infile.ReadByte();
            if (x == '#')
            {
                ++i;
                do
                {
                    x = (char)infile.ReadByte();
                    ++i;
                }
                while (x != '#');
            }
            else
                if (x != ' ')
                    outfile.WriteByte((byte)char.ToUpper(x));
        }
        infile.Close();
        outfile.Close();
    }

    public static void scanner()
    {
        FileStream infile;
        int tam;
        int kl = 1;
        char x;
        String aux="";

        infile = new System.IO.FileStream("pftmp.txt",
                                          System.IO.FileMode.Open,
                                          System.IO.FileAccess.Read);

        MeuCompiladorDAL.deletaTTokensValidos();
        tam = (int)infile.Length;
        for (int i = 0; i < tam; ++i)
        {
```

```
    x = (char)infile.ReadByte();

    if (char.IsDigit(x))
    {
        while (char.IsDigit(x))
        {
            aux = aux + x;
            x = (char)infile.ReadByte();
            ++i;
        }
        Token.setCodigo("200");
        Token.setToken(aux);
        Token.setTipo("Inteiro");
        Token.setLinha(kl.ToString());
        MeuCompiladorDAL.inseriUmTokenValido();
        aux = "";
    }

    if (char.IsLetter(x))
    {
        aux = "";
        while (char.IsLetter(x))
        {
            aux = aux + x;
            x = (char)infile.ReadByte();
            ++i;
        }
        Token.setToken(aux);
        Token.setTipo("String");
        Token.setLinha(kl.ToString());
        MeuCompiladorDAL.consultaUmToken();
        if (Erro.getErro())
            return;
        else
            MeuCompiladorDAL.inseriUmTokenValido();
        aux = "";
    }

    if (char.IsPunctuation(x) || char.IsSymbol(x))
    {
        Token.setToken(x.ToString());
        Token.setTipo("Delimitador");
        Token.setLinha(kl.ToString());
        MeuCompiladorDAL.consultaUmToken();
        if (Erro.getErro())
            return;
        else
            MeuCompiladorDAL.inseriUmTokenValido();
    }

    if (x == 13) ++kl;
}
infile.Close();
```

```
 class AnalisadorSintaticoBLL
    {
        public static void validaSequencia()
        {
            Erro.setErro(false);
            MeuCompiladorDAL.populaGabarito(int.Parse(Token.getCodigo()));
            MeuCompiladorDAL.leGabarito();
            if (Erro.getErro())
            {
                Erro.setErro("Linha " + Token.getLinha() + "- token inesperado: " +
Token.getToken());
                return;
            }

            while (!Erro.getErro())
            {
                if (Token.getCodigo() != Gabarito.getInfo())
                {
                    Erro.setErro("Linha " + Token.getLinha() + "- token inesperado: "
+ Token.getToken());
                    return;
                }
                if (Gabarito.getNext() != "eof") MeuCompiladorDAL.leUmTokenValido();
                MeuCompiladorDAL.leGabarito();
            }
            Erro.setErro(false);
        }

        public static void analiseSintatica()
        {

            MeuCompiladorDAL.populaDR();

            MeuCompiladorDAL.leUmTokenValido();
            while (Erro.getErro() == false)
            {
                validaSequencia();
                if (Erro.getErro()) return;
                MeuCompiladorDAL.leUmTokenValido();
            }
            Erro.setErro(false);
        }

    }
```

```
    class MeuCompiladorDAL
    {
        private static String strConexao = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=Compilador.mdb";
        private static OleDbConnection conn = new OleDbConnection(strConexao);
        private static OleDbCommand strSQL;
```

```
        private static OleDbDataReader resultAS, resultgab, resultqtdarg,
resultlimites;

        public static void conecta()
        {
            Erro.setErro(false);
            try
            {
                conn.Open();
            }
            catch
            {
                Erro.setErro("A conexão falhou!");
                return;
            }
        }

        public static void desconecta()
        {
            conn.Close();
        }

        public static void inseriUmTokenValido()
        {
            String aux = "insert into TTokensValidos(codigo,token,tipo,linha) values
(" + Token.getCodigo() + ",'" + Token.getToken() + "','" + Token.getTipo() + "'," +
Token.getLinha() + ")";

            strSQL = new OleDbCommand(aux, conn);
            strSQL.ExecuteNonQuery();
        }

        public static void deletaTTokensValidos()
        {
            String aux = "delete * from TTokensValidos";

            strSQL = new OleDbCommand(aux, conn);
            strSQL.ExecuteNonQuery();
        }

        public static void deletaDelimitadores()
        {
            String aux = "delete * from TTokensValidos where Tipo = 'Delimitador'";

            strSQL = new OleDbCommand(aux, conn);
            strSQL.ExecuteNonQuery();
        }

        public static void consultaUmToken()
        {
            OleDbDataReader result;
            String aux = "select * from TTokens where Token ='" + Token.getToken() +
"'";
```

```
            strSQL = new OleDbCommand(aux, conn);
            result = strSQL.ExecuteReader();
            Erro.setErro(false);
            if (result.Read())
                Token.setCodigo("" + result.GetInt32(0));
            else
                Erro.setErro("Linha " + Token.getLinha() + ": " + Token.getToken() + "
(Token não identificado)");
        }

        public static void populaDR()
        {
            String aux = "select * from TTokensValidos";

            strSQL = new OleDbCommand(aux, conn);
            resultAS = strSQL.ExecuteReader();
        }

        public static void leUmTokenValido()
        {
            Erro.setErro(false);
            if (resultAS.Read())
            {
                Token.setCodigo("" + resultAS.GetInt32(0));
                Token.setToken(resultAS.GetString(1));
                Token.setTipo(resultAS.GetString(2));
                Token.setLinha("" + resultAS.GetInt32(3));
            }
            else
                Erro.setErro(true);

        }

        public static void populaGabarito(int _code)
        {
            String aux = "select * from gabarito where code = " + _code;

            strSQL = new OleDbCommand(aux, conn);
            resultgab = strSQL.ExecuteReader();
        }

        public static void leGabarito()
        {
            Erro.setErro(false);
            if (resultgab.Read())
            {
                Gabarito.setCode("" + resultgab.GetInt32(0));
                Gabarito.setPrior(resultgab.GetString(1));
                Gabarito.setInfo("" + resultgab.GetInt32(2));
                Gabarito.setNext(resultgab.GetString(3));
            }
            else
                Erro.setErro(true);
        }
```

```
        public static int leQtdArgumentos()
        {
            String aux = "select * from TQtdArgumentos where Codigo =" +
Token.getCodigo();

            strSQL = new OleDbCommand(aux, conn);
            resultqtdarg = strSQL.ExecuteReader();
            resultqtdarg.Read();
            return resultqtdarg.GetInt32(1);
        }

        public static void leUmLimite()
        {
            String aux = "select * from TArgLimites where codigo =" +
ArgLim.getCodigo() + " and posicao =" + ArgLim.getposicao();

            strSQL = new OleDbCommand(aux, conn);
            resultlimites = strSQL.ExecuteReader();
            if (resultlimites.Read())
            {
                ArgLim.setminimo("" + resultlimites.GetInt32(2));
                ArgLim.setmaximo("" + resultlimites.GetInt32(3));
            }
        }

    }
```