

**CENTRO *UNIVERSITÁRIO* DE BARRA MANSA  
PRÓ-REITORIA ACADÊMICA**

**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

**SISTEMA RECONHECEDOR DE  
CONSTANTES NUMÉRICAS EM  
FORTRAN**

Por:  
Leniel Braz de Oliveira Macaferi  
Wellington Magalhães Leite

**Barra Mansa  
7 de Março de 2006**

**CENTRO *UNIVERSITÁRIO* DE BARRA MANSA  
PRÓ-REITORIA ACADÊMICA**

**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

**SISTEMA RECONHECEDOR DE  
CONSTANTES NUMÉRICAS EM  
FORTRAN**

Por:  
Leniel Braz de Oliveira Macaferi  
Wellington Magalhães Leite

Trabalho acadêmico apresentado à disciplina de Construção de Compiladores do Curso de Engenharia de Computação do Centro Universitário de Barra Mansa como requisito parcial de avaliação da nota 1, sob a orientação do professor José Nilton Cantarino Gil.

**Barra Mansa  
7 de Março de 2006**

## Sumário

	Pág.
1. Introdução.....	5
1.1 Objetivo .....	5
1.2 Definição.....	5
2. Desenvolvimento .....	6
2.1 Mapeamento das constantes .....	6
2.2 Mapeamento das funções.....	6
2.3 Mapeamento da chamada principal .....	10
3. Aplicação .....	12
3.1 Validação de expressões .....	12
3.1.1 Uma única expressão .....	13
3.1.2 Uma lista de expressões .....	14
4. Conclusão.....	15
5. Bibliografia.....	16
6. Anexo .....	17

## **Resumo**

Um método muito didático para a construção do front-end de um compilador é aquele fundamentado em diagramas de transição. Logo, seu estudo ajuda a familiarização com as atividades envolvidas no projeto de um compilador. Esse trabalho apresenta um sistema reconhecedor de constantes numéricas em FORTRAN. O mesmo é desenvolvido a partir de um diagrama de transição de estados e sua codificação segue os padrões e moldes da linguagem de programação C#.

Palavras-chave: compilador, diagrama de transição, sistema, linguagem C#.

## 1. Introdução

### 1.1 Objetivo

O nosso objetivo no presente trabalho é implementar um reconhecedor de constantes numéricas referente à linguagem FORTRAN utilizando a linguagem de programação C#. O mesmo deve atender às especificações de um diagrama de transição de estados. Para tanto, criaremos um projeto do tipo console no ambiente de desenvolvimento Microsoft Visual C# 2005 Express Edition.

### 1.2 Definição

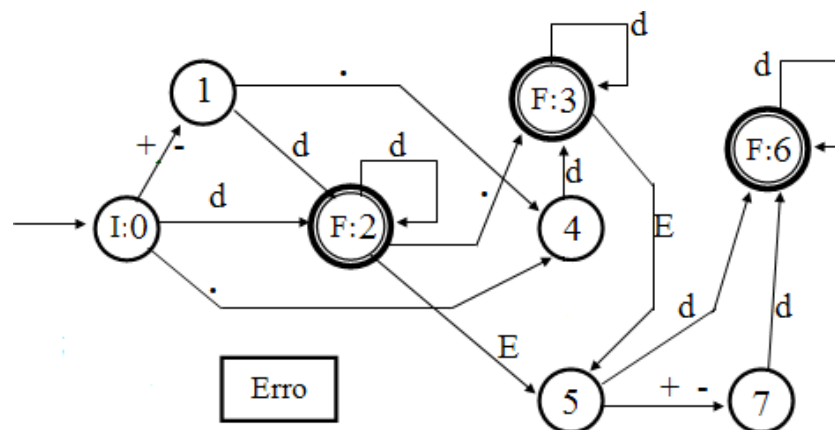


Figura 1: diagrama de transição de estados reconhecedor de constantes numéricas em FORTRAN.

Alfabeto  $\Sigma = \{d, +, -, ., E\}$  (d é um dígito qualquer)

Os arcos de qualquer vértice para o vértice Erro não são mostrados, mas existem.

São reconhecidas expressões como: 13, +13, -13, 13., 1.25, .25, -.25, -32.43, 13E-15, 13.E-15, -13.25E+72, .75E5, etc.

Não são reconhecidas expressões como: ++13, .E13, etc.

O vértice de valor 0 é o estado inicial.

Os vértices com círculos mais fortes são estados finais.

## 2. Desenvolvimento

### 2.1 Mapeamento das constantes

```
/* Enumeração, onde cada item corresponde a um estado do diagrama de
transição. */
```

```
public enum possibleStates
{
    s0 = 0,
    s1,
    s2,
    s3,
    s4,
    s5,
    s6,
    s7,
    s8,
    err
}
```

```
/* Vetor do tipo possibleStates (estados possíveis), o qual contém os
estados finais aceitos pelo diagrama de transição. */
```

```
public static possibleStates[] finalStates = { possibleStates.s2,
possibleStates.s3, possibleStates.s6 };
```

### 2.2 Mapeamento das funções

```
/// <sumário>
/// Verifica se o último estado atingido pela máquina é pertencente ao
vetor de estados finais.
/// </sumário>
```

```
public static bool IsFinalState(possibleStates currentState)
{
    for(int i = 0; i < finalStates.Length; i++)
        if(currentState == finalStates[i])
            return true;

    return false;
}
```

```
/// <sumário>
/// Reconhece o estado atual e o caractere "label" sendo analisado, valores
esses passados como parâmetro. Em seguida a função faz a transição de
```

estado caso seja satisfeita alguma condição, caso contrário, a função retornará um sinalizador de erro.

```
///  
//</sumário>
```

```
public static possibleStates Recognizer(possibleStates currentState, char  
c)  
{  
    switch(currentState)  
    {  
        case possibleStates.s0:  
        {  
            if(c == '+' || c == '-')  
                return possibleStates.s1;  
  
            if(char.IsDigit(c))  
                return possibleStates.s2;  
  
            if(c == '.')  
                return possibleStates.s4;  
  
            break;  
        }  
  
        case possibleStates.s1:  
        {  
            if(char.IsDigit(c))  
                return possibleStates.s2;  
  
            if(c == '.')  
                return possibleStates.s4;  
  
            break;  
        }  
  
        case possibleStates.s2:  
        {  
            if(char.IsDigit(c))  
                return possibleStates.s2;  
  
            if(c == '.')  
                return possibleStates.s3;  
  
            if(c == 'E')  
                return possibleStates.s5;  
  
            break;  
        }  
  
        case possibleStates.s3:  
        {  
            if(char.IsDigit(c))  
                return possibleStates.s3;  
  
            if(c == 'E')  
                return possibleStates.s5;  
  
            break;  
        }  
  
        case possibleStates.s4:  
        {
```

```

        if(char.IsDigit(c))
            return possibleStates.s3;

        break;
    }

    case possibleStates.s5:
    {
        if(char.IsDigit(c))
            return possibleStates.s6;

        if(c == '+' || c == '-')
            return possibleStates.s7;

        break;
    }

    case possibleStates.s6:
    {
        if(char.IsDigit(c))
            return possibleStates.s6;

        break;
    }

    case possibleStates.s7:
    {
        if(char.IsDigit(c))
            return possibleStates.s6;

        break;
    }
    }
    return possibleStates.err;
}

/// <sumário>
/// Lê uma expressão, reconhece seus caracteres, muda os estados da máquina
de acordo com esses caracteres e então valida a entrada.
/// </sumário>

public static void SingleExpression()
{
    do
    {
        // A máquina aponta para o estado inicial do diagrama de transição de
estados.
        possibleStates currentState = possibleStates.s0;

        Console.WriteLine("\n\nEnter the expression to be evaluated: ");

        // strExpression recebe a expressão entrada pelo usuário.
        string strExpression = Console.ReadLine();

        /* Para cada caractere (label) da expressão, chama a função
Recognizer (Reconhedora) que por sua vez muda o estado da máquina de
acordo com os label. */
        for(int i = 0; strExpression.Length > i; ++i)
            if(currentState != possibleStates.err)
                currentState = Recognizer(currentState, strExpression[i]);
            else

```

```

        break;

        /* Chama a função IsFinalState (ÉEstadoFinal?) para verificar se o
estado onde a máquina parou é final ou não. */
        if(IsFinalState(currentState))
            Console.WriteLine("\n Valid expression.\n");
        else
            Console.WriteLine("\n Invalid expression!\n");

        Console.Write("Do you wanna try again? (y\n) ");
    }
    while(Console.ReadKey().KeyChar == 'y');
}

/// <sumário>
/// Lê um arquivo, reconhece suas linhas, expressão por expressão e muda os
estados da máquina de acordo com cada expressão. Em outras palavras, valida
toda a lista.
/// </sumário>

public static void SetOfExpressions()
{
    do
    {
        Console.Write("\n\nEnter the file path: ");

        // Obtém o nome do arquivo.
        string fileName = Console.ReadLine();

        // Verifica se o arquivo existe.
        if(!File.Exists(fileName))
            Console.Write("\n File not found!\n\n");
        else
        {
            // Lê e armazena todas as linhas do arquivo.
            StreamReader sReader = new StreamReader(fileName);

            string expression;

            // Avalia o conteúdo de cada linha até atingir o fim do arquivo.
            while((expression = sReader.ReadLine()) != null)
            {

                // A máquina aponta para o estado inicial do diagrama de
transição de estados.
                possibleStates currentState = possibleStates.s0;

                /* Para cada caractere (label) da expressão, chama a função
Recognizer (Reconhecadora) que por sua vez muda o estado da máquina de
acordo com os label. */
                for(int i = 0; expression.Length > i; ++i)
                    if(currentState != possibleStates.err)
                        currentState = Recognizer(currentState, expression[i]);
                    else
                        break;

                /* Chama a função IsFinalState (ÉEstadoFinal?) para verificar se
o estado onde a máquina parou é final ou não. */
                if(IsFinalState(currentState))
                    Console.WriteLine("\n{0} is a valid expression.", expression);
                else

```

```

        Console.WriteLine("\n{0} is an invalid expression!",
expression);
    }
    sReader.Close();
}
Console.Write("\nDo you wanna try again? (y\n) ");
}
while(Console.ReadKey().KeyChar == 'y');
}

```

### 2.3 Mapeamento da chamada principal

```

public static void Main(string[] args)
{
    Console.Title = "State Transition System for recognizing numeric
constants in FORTRAN";

    Console.BackgroundColor = ConsoleColor.White;
    Console.ForegroundColor = ConsoleColor.Black;

    char ch;

    do
    {
        Console.Clear();

        // Imprime um anúncio
        Console.Write("\nState Transition System C# Sample Application\n");
        Console.Write("Copyright ©2006 Leniel Braz de Oliveira Macaferi &
Wellington Magalhães Leite.\n\n");
        Console.Write("UBM COMPUTER ENGINEERING - 7TH SEMESTER
[http://www.ubm.br/]\n\n");

        // Descreve a função do sistema
        Console.Write("This program example demonstrates the State Transition
Diagram's algorithm for\n");
        Console.Write("numeric constants validation in FORTRAN.\n\n");

        // Descreve as opções do sistema
        Console.Write("You can validate expressions by two different ways as
follow:\n\n");
        Console.Write("1 - A single expression by providing an entry.\n\n");
        Console.Write("2 - A set of expressions by providing an input
file.\n");
        Console.Write(" * Notice: the expressions must be separated in-
lines.\n");

        Console.Write("\n\nEnter your choice: ");

        ch = Console.ReadKey().KeyChar;

        switch(ch)
        {
            case '1':
            {
                SingleExpression();
                break;
            }
            case '2':
            {
                SetOfExpressions();
            }
        }
    }
}

```

```
        break;
    }
}
while(ch == '1' || ch == '2');
}
```

### **3. Aplicação**

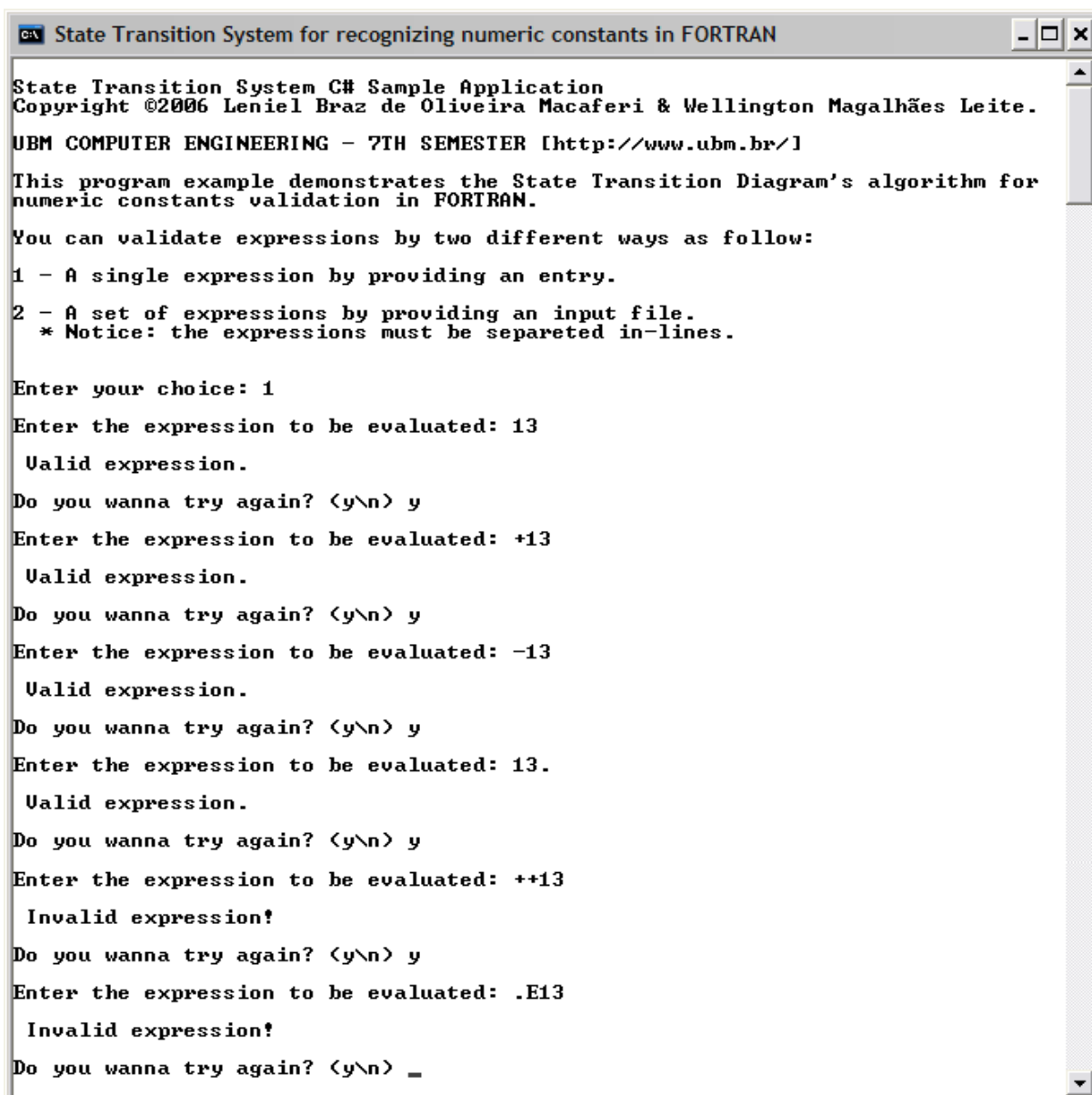
#### **3.1 Validação de expressões**

Validações realizadas através das expressões fornecidas pelo usuário com o auxílio da entrada e saída padrão do computador, ou seja, teclado e monitor.

A validação é realizada com o fornecimento de uma única expressão ou com o fornecimento de um arquivo que contenha expressões separadas em linhas.

### 3.1.1 Uma única expressão

Valores de constantes descritas na seção 1.2 digitadas uma de cada vez.



```
C:\ State Transition System for recognizing numeric constants in FORTRAN

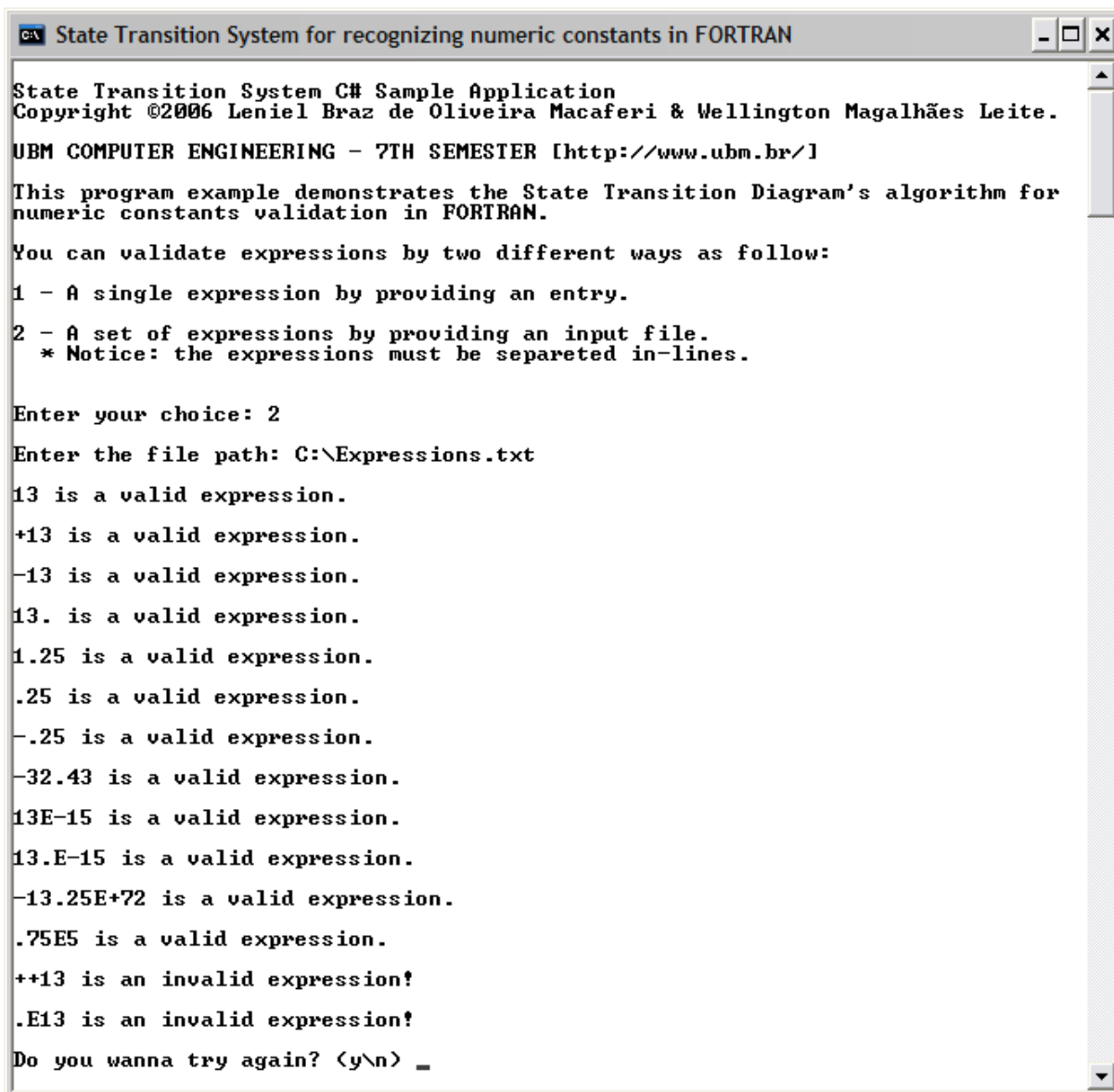
State Transition System C# Sample Application
Copyright ©2006 Leniel Braz de Oliveira Macaferi & Wellington Magalhães Leite.
UBM COMPUTER ENGINEERING - 7TH SEMESTER [http://www.ubm.br/]
This program example demonstrates the State Transition Diagram's algorithm for
numeric constants validation in FORTRAN.
You can validate expressions by two different ways as follow:
1 - A single expression by providing an entry.
2 - A set of expressions by providing an input file.
* Notice: the expressions must be separeted in-lines.

Enter your choice: 1
Enter the expression to be evaluated: 13
Valid expression.
Do you wanna try again? (y\n) y
Enter the expression to be evaluated: +13
Valid expression.
Do you wanna try again? (y\n) y
Enter the expression to be evaluated: -13
Valid expression.
Do you wanna try again? (y\n) y
Enter the expression to be evaluated: 13.
Valid expression.
Do you wanna try again? (y\n) y
Enter the expression to be evaluated: ++13
Invalid expression!
Do you wanna try again? (y\n) y
Enter the expression to be evaluated: .E13
Invalid expression!
Do you wanna try again? (y\n) _
```

Figura 2: valores de constantes descritas na seção 1.2 digitadas uma de cada vez.

### 3.1.2 Uma lista de expressões

Valores de constantes descritas na seção 1.2 fornecidas por um arquivo.



```
State Transition System C# Sample Application
Copyright ©2006 Leniel Braz de Oliveira Macaferi & Wellington Magalhães Leite.
UBM COMPUTER ENGINEERING - 7TH SEMESTER [http://www.ubm.br/]
This program example demonstrates the State Transition Diagram's algorithm for
numeric constants validation in FORTRAN.
You can validate expressions by two different ways as follow:
1 - A single expression by providing an entry.
2 - A set of expressions by providing an input file.
  * Notice: the expressions must be separeted in-lines.

Enter your choice: 2
Enter the file path: C:\Expressions.txt
13 is a valid expression.
+13 is a valid expression.
-13 is a valid expression.
13. is a valid expression.
1.25 is a valid expression.
.25 is a valid expression.
-.25 is a valid expression.
-32.43 is a valid expression.
13E-15 is a valid expression.
13.E-15 is a valid expression.
-13.25E+72 is a valid expression.
.75E5 is a valid expression.
++13 is an invalid expression!
.E13 is an invalid expression!
Do you wanna try again? <y\n> _
```

Figura 3: valores de constantes descritas na seção 1.2 fornecidas por um arquivo.

#### **4. Conclusão**

Com o desenvolvimento deste trabalho, tivemos a oportunidade de dar um passo importante em nosso estudo sobre construção de compiladores, mesmo sendo um caso com diagrama de transição léxica apresentando poucos estados (obs.: diagramas de transições léxicas são autômatos finitos).

Um dos pontos interessantes foi o fato de podermos explorar as potencialidades de uma linguagem de programação, em nosso caso a linguagem de programação C#, a fim de elaborarmos um programa eficiente. Assim, entramos em contato com novos recursos ou recursos que até então não havíamos utilizado.

A respeito do programa elaborado, todos os testes feitos corresponderam ao respectivo diagrama de transição léxica para reconhecer constantes numéricas em FORTRAN, conforme apresentado no item aplicação.

O mais importante é que este trabalho nos dá uma visão geral de como precisaremos proceder em outros casos mais complexos, pois nos ajuda na familiarização com as atividades envolvidas no projeto de um compilador. Assim, poderemos dominar os conceitos na área de construção de compiladores o que é de grande valia para um Engenheiro de Computação.

## **5. Bibliografia**

ROQUE, K. (tradução). Microsoft® C# Segredos da Linguagem. 1ª ed. Rio de Janeiro: Campus, 2001.

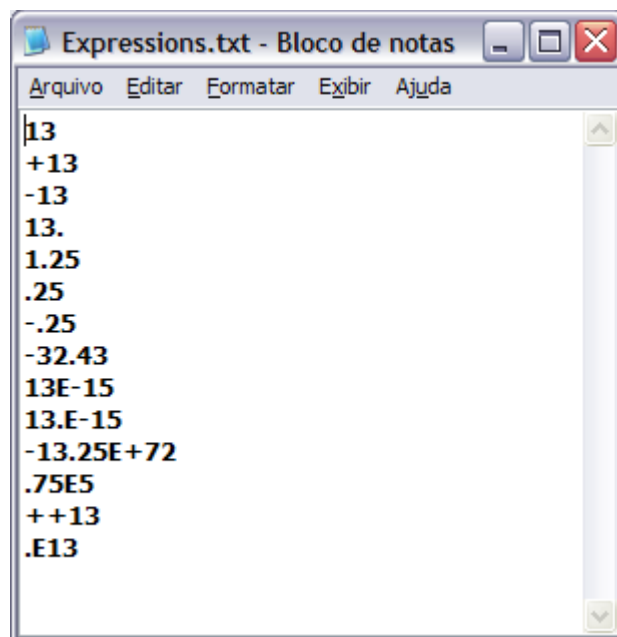
Mokarzel, F. C. - Notas de aula sobre Compiladores.  
<http://www.comp.ita.br/pessoas/professores/fabio.htm>

Microsoft Visual C# 2005 Express Edition  
<http://msdn.microsoft.com/vstudio/express/visualcsharp/default.aspx>

Sites disponíveis em: 28/2/2006

## 6. Anexo

Expressions.txt



Anexo 1: arquivo usado para armazenar os valores de constantes da seção 1.2.