unibh

CENTRO UNIVERSITÁRIO DE BELO HORIZONTE

Trabalho Prático

Stella Oliveira Moreira

Belo Horizonte – 2018

Índice

Introdução

O propósito desse trabalho é desenvolver um compilador na linguagem C# que é uma linguagem de programação orientada a objetos disponível gratuitamente para todo público.

Um compilador é um programa, que a partir de um código fonte escrito em uma linguagem compilada, é criado um programa semanticamente equivalente, porém escrito em outra linguagem. De forma geral, um compilador traduz um programa de uma linguagem sendo possível ser entendida por uma pessoa.

Esse projeto em questão apresenta a Analise Léxica, que se trata de um processo de análise de entrada de linhas de caracteres (código fonte) que produz uma sequência de símbolos (léxicos), que podem ser manipulados mais facilmente por um *parser* (leitor de saída). A análise léxica verifica o alfabeto a partir de uma tabela de símbolos, se existe ou não algum caractere que faz parte dessa tabela.

1. A entrada é lida, uma de cada vez mudando o estado em que os caracteres se encontram. Quando o analisador encontra um caractere que ele não identifica como correto, ele chama o “estado morto” logo, volta a última analise aceita e tem o tipo de comprimento do léxico valido.
2. Os caracteres são repassados do léxico para produzir um valor, o tipo do léxico combinado com seu valor é adequado o que pode ser dado a um parser.

Desenvolvimento

A linguagem C# possui uma lógica hierárquica, constitui desde as operações básicas (adição, multiplicação, subtração) e soma às mais complicadas e únicas como encapsulamentos e sobrecargas múltiplas. No C# não existe herança múltipla, ou seja, cada classe só pode herdar uma e só uma classe, mantendo assim o código e a gestão de memória organizada. Possui também uma gestão de memória única denominada *coletor de lixo*, que recicla a memória e obtém uma maior eficiência.

No que diz respeito a passagem de parâmetros, no C# existem dois tipos de passagem, nomeadamente, passagem por valor e por referência. Na passagem por valor é feita uma cópia do argumento para o parâmetro do mesmo, assim garantimos que a alteração efetuada dentro do parâmetro não tem valor do argumento externo. Assim mantemos a integridade do argumento inicial sempre *saudável*. Em passagem por referência, podemos dizer que é o oposto da passagem por valor, sendo mais utilizada, o valor dentro e fora do método é alterado.

Desempenho Global:

No desenvolvimento deste trabalho foram efetuadas várias pesquisas sobre o tema (Análise léxica com uso de tabela de símbolos), e a busca para um código bem estruturado foi um dos requisitos para o desenvolvimento do mesmo.

Fases de Elaboração:

1. Pesquisa
2. Desempenho
3. Esquematização
4. Desenvolvimento do autômato
5. Implementação

ID etapa	Descrição de Etapa	ID próxima etapa	Tipo de forma	Data de Início	Data de Término
P100	Início	P200	Início		
P200	Pesquisa	P300	Processo	02/04/2018	04/04/2018
P300	Esquematização	P400	Documento	04/04/2018	
P400	Desenvolvimento do Autômato	P500	Decisão	04/04/2018	04/04/2018
P500	Implementação	P600	Subprocesso	05/04/2018	08/04/2018
P600	Termino		Termino	08/04/2018	

Explicação do código

Todo o código foi desenvolvido com base no exemplo de Lexer disponibilizado pelo professor, o código implementado em busca de otimização na leitura dos caracteres, toda leitura do arquivo foi feita byte a byte.

No uso do código é necessário especificar o caminho do arquivo (arquivo.txt) na classe “AnalisadorLex.cs” na região especificada (Leitura de arquivo).

```
#region Leitura de Arquivo
arquivo = new FileStream("E:\\StellaOliveiraMoreira\
  \TrabalhoPratico01\\Arquivo_Teste_Erro\\Teste03_Erro.txt",
  FileMode.Open);
#endregion
```

Foram feitos diversos testes para que a implementação tenha sido um sucesso, e conforme solicitado no contexto do trabalho foram feitos três testes onde não ocorre nenhum erro e três testes onde possuem erros.

OBS: O código fonte de cada teste está em anexo ao arquivo.

Testes com sucesso:

```
------------------------
Token: <KW , "program ">
Linha: 1        Coluna: 1
------------------------
Token: <ID , "p1 ">
Linha: 1        Coluna: 2
------------------------
Token: <SMB_OBC , "{ ">
Linha: 2        Coluna: 1
------------------------
Token: <KW , "num ">
Linha: 3        Coluna: 1
------------------------
Token: <OP_ASS , "= ">
Linha: 3        Coluna: 2
------------------------
Token: <NUM_CONST , "1 ">
Linha: 3        Coluna: 3
------------------------
Token: <SMB_SEM , "; ">
Linha: 3        Coluna: 4
------------------------
Token: <KW , "if ">
Linha: 4        Coluna: 1
------------------------
Token: <SMB_OPA , "( ">
Linha: 4        Coluna: 2
------------------------
Token: <KW , "num ">
Linha: 4        Coluna: 3
------------------------
Token: <OP_GT , "> ">
Linha: 4        Coluna: 4
------------------------
Token: <NUM_CONST , "0 ">
Linha: 4        Coluna: 5
------------------------
Token: <SMB_CPA , ") ">
Linha: 4        Coluna: 6
------------------------
Token: <SMB_OBC , "{ ">
Linha: 4        Coluna: 7
------------------------
Token: <KW , "write ">
Linha: 5        Coluna: 1
------------------------
Token: <STRING , "numero > 0 ">
Linha: 5        Coluna: 2
------------------------
Token: <SMB_SEM , "; ">
Linha: 5        Coluna: 3
------------------------
Token: <SMB_CBC , "} ">
Linha: 6        Coluna: 1
------------------------
Token: <SMB_CBC , "} ">
Linha: 7        Coluna: 1
------------------------
Token: <EOF , "EOF ">
Linha: 7        Coluna: 1


                Arquivo Finalizado
```

```
------------------------
Token: <KW , "program ">
Linha: 1        Coluna: 1
------------------------
Token: <SMB_OPA , "( ">
Linha: 1        Coluna: 2
------------------------
Token: <KW , "num ">
Linha: 1        Coluna: 3
------------------------
Token: <ID , "var_2 ">
Linha: 1        Coluna: 4
------------------------
Token: <SMB_COM , ", ">
Linha: 1        Coluna: 5
------------------------
Token: <KW , "num ">
Linha: 1        Coluna: 6
------------------------
Token: <ID , "var_1 ">
Linha: 1        Coluna: 7
------------------------
Token: <SMB_CPA , ") ">
Linha: 1        Coluna: 8
------------------------
Token: <SMB_OBC , "{ ">
Linha: 2        Coluna: 1
------------------------
Token: <KW , "if ">
Linha: 3        Coluna: 1
------------------------
Token: <SMB_OPA , "( ">
Linha: 3        Coluna: 2
------------------------
Token: <ID , "var_2 ">
Linha: 3        Coluna: 3
------------------------
Token: <OP_NE , "!= ">
Linha: 3        Coluna: 4
------------------------
Token: <ID , "var_1 ">
Linha: 3        Coluna: 5
------------------------
Token: <SMB_CPA , ") ">
Linha: 3        Coluna: 6
------------------------
Token: <SMB_OBC , "{ ">
Linha: 3        Coluna: 7
------------------------
Token: <KW , "write ">
Linha: 4        Coluna: 1
------------------------
Token: <STRING , "é diferente ">
Linha: 4        Coluna: 2
------------------------
Token: <SMB_SEM , "; ">
Linha: 4        Coluna: 3
------------------------
Token: <SMB_CBC , "} ">
Linha: 5        Coluna: 1
------------------------
Token: <EOF , "EOF ">
Linha: 5        Coluna: 1


                Arquivo Finalizado
```

```
------------------------
Token: <KW , "program ">
Linha: 1        Coluna: 1
------------------------
Token: <SMB_OPA , "( ">
Linha: 1        Coluna: 2
------------------------
Token: <SMB_CPA , ") ">
Linha: 1        Coluna: 3
------------------------
Token: <SMB_OBC , "{ ">
Linha: 2        Coluna: 1
------------------------
Token: <STRING , "String ">
Linha: 3        Coluna: 1
------------------------
Token: <ID , "numero ">
Linha: 4        Coluna: 1
------------------------
Token: <OP_ASS , "= ">
Linha: 4        Coluna: 2
------------------------
Token: <NUM_CONST , "123 ">
Linha: 4        Coluna: 3
------------------------
Token: <SMB_SEM , "; ">
Linha: 4        Coluna: 4
------------------------
Token: <SMB_CBC , "} ">
Linha: 5        Coluna: 1
------------------------
Token: <EOF , "EOF ">
Linha: 5        Coluna: 1


                Arquivo Finalizado
```

Testes com falha:

```
--------------------------
Token: <KW , "public ">
Linha: 1        Coluna: 1
--------------------------
Token: <ID , "static ">
Linha: 1        Coluna: 2
--------------------------
Token: <ID , "void ">
Linha: 1        Coluna: 3
--------------------------
Token: <ID , "main ">
Linha: 1        Coluna: 4
--------------------------
Token: <SMB_OPA , "( ">
Linha: 1        Coluna: 5
--------------------------
Token: <SMB_CPA , ") ">
Linha: 1        Coluna: 5

        Erro Lexico:
--------------------------
Erro: String deve ser fechada com } antes do fim de arquivo


                Arquivo Finalizado
```

```
-----------------------
Token: <KW , "public ">
Linha: 1        Coluna: 1
-----------------------
Token: <ID , "static ">
Linha: 1        Coluna: 2
-----------------------
Token: <ID , "void ">
Linha: 1        Coluna: 3
-----------------------
Token: <ID , "main ">
Linha: 1        Coluna: 4
-----------------------
Token: <SMB_OPA , "( ">
Linha: 1        Coluna: 5
-----------------------
Token: <SMB_CPA , ") ">
Linha: 1        Coluna: 5
-----------------------
Token: <SMB_OBC , "{ ">
Linha: 1        Coluna: 6

       Erro Lexico:
-----------------------
Erro: String deve ser fechada com " antes do fim de arquivo


             Arquivo Finalizado
```

```
-----------------------
Token: <KW , "public ">
Linha: 1        Coluna: 1
-----------------------
Token: <ID , "static ">
Linha: 1        Coluna: 2
-----------------------
Token: <ID , "void ">
Linha: 1        Coluna: 3
-----------------------
Token: <ID , "main ">
Linha: 1        Coluna: 4

       Erro Lexico:
-----------------------
Erro: String deve ser fechada com ) antes do fim de arquivo


             Arquivo Finalizado
```

unibh

Classe AnaliseLex

```
// Construtor abre o arquivo de: xxxxx e verifica
public AnalisadorLex(String arquivoEntrada)
{
    tabelaSimbolos = new TabelaSimbolos();
    try
    {
        #region Leitura de Arquivo
        arquivo = new FileStream("E:\\StellaOliveiraMoreira\
          \TrabalhoPratico01\\Arquivo_Teste_Erro\\Teste03_Erro.txt",
          FileMode.Open);
        #endregion
    }
    catch (IOException excep)
    {
        Console.WriteLine("Erro de abertura do arquivo " +
          arquivoEntrada + "\nException: " + excep);
        Environment.Exit(1);
        Console.ReadLine();
    }
    catch (Exception excep)
    {
        Console.WriteLine("Erro do programa ou falha da tabela de
          símbolos \nException: " + excep);
        Environment.Exit(2);
    }
}
```

Método responsável por analisar o arquivo

```
//Fecha o arquivo aberto
public void fechaArquivo()
{
    try
    {
        arquivo.Close();
    }
    catch (IOException errorFile)
    {
        Console.WriteLine("Erro ao fechar o arquivo \nException: " +
          errorFile);
```

\Área de Trabalho\Finalizado01\AnalisadorLex.cs 2

```
        Environment.Exit(3);
        Console.ReadLine();
    }
}
```

Método responsável por fechar o arquivo

```
//Sinaliza erro do token
public void sinalizaErro(String mensagem)
{
    Console.WriteLine("\n\tErro Lexico: " + mensagem + "\n");
    fechaArquivo();
    Console.WriteLine("\n\n\t\tArquivo Finalizado");
    Console.ReadLine();
}
```

Método responsável por sinalizar o erro

```
//Retorna uma posição do buffer de leitura
public void retornaPonteiro()
{
    try
    {
        if (lookahead != END_OF_FILE)
        {
            //arquivo.Seek(arquivo.Position - 1, SeekOrigin.Current);
            arquivo.Position--;

        }
    }
    catch (IOException excep)
    {
        Console.WriteLine("Falha ao retornar a Leitura \nException: "
          + excep);
        Environment.Exit(3);
        Console.ReadLine();
    }
}
```

Método responsável por retornar à posição anterior lida.

```
public Token proximoToken()
{
    var lexema = new StringBuilder();
    int estado = 1;
    char caracter;

    for (var i = 0; i <= arquivo.Length; i++)
    {
        caracter = '\u0000'; //Caracter Null

        //Avança
        try
        {
            lookahead = arquivo.ReadByte();
            if (lookahead != END_OF_FILE)
            {
                caracter = (char)lookahead;
            }
        }
```

Método responsável por analisar o próximo token

```
public static void Main(string[] args)
{
    //Parametro de arquivo
    AnalisadorLex lexer = new AnalisadorLex("Teste03_Erro.txt");
    Token token;
    TabelaSimbolos tabelaSimbolos = new TabelaSimbolos();

    //Enquanto não houver erros e nem for fim de arquivo:
    do
    {
        token = lexer.proximoToken();

        //Imprime o Token
        if (token != null)
        {
            Console.WriteLine("-------------------------\n" +
                              "Token: " + token.ToString());
        }
    } while (lookahead != END_OF_FILE);
    lexer.fechaArquivo();
    Console.WriteLine("\n\n\t\tArquivo Finalizado");
    Console.ReadLine();
}
```

Método principal, responsável por direcionar ações no decorrer do programa

```
public class Token
{
    private string lexema;
    private EnumTab classe;
    public int nLinha;
    public int nColuna;

    //Construtor
    public Token(EnumTab classe, String lexema, int linha, int Coluna)
    {
        this.classe = classe;
        this.lexema = lexema;
        this.nLinha = linha;
        this.nColuna = Coluna;
    }

    public String getLexema()
    {

        return lexema;
    }

    public void setLexema(String lexema)
    {

        this.lexema = lexema;
    }
    public EnumTab getClasse()
    {
```

Classe responsável por retornar o token encontrado

```
public override string ToString()
{
    return "<" + classe + " , \"" + lexema + " \">\nLinha: " + getLinha
      () + " \tColuna: " + getColuna();
}
```

Método responsável por retornar a estrutura do token encontrado (Token, Lexema, Linha e Coluna)

```
public class TabelaSimbolos
{
    private Dictionary<Token, InfIdentificador> tabelaSimbolos;

    //Construtor
    public TabelaSimbolos()
    {
        tabelaSimbolos = new Dictionary<Token, InfIdentificador>();

        //Palavras reservadas
        Token palavra;

        palavra = new Token(EnumTab.KW, "program", 0, 0);
        tabelaSimbolos[palavra] = new InfIdentificador();

        palavra = new Token(EnumTab.KW, "if", 0, 0);
        tabelaSimbolos[palavra] = new InfIdentificador();

        palavra = new Token(EnumTab.KW, "else", 0, 0);
        tabelaSimbolos[palavra] = new InfIdentificador();

        palavra = new Token(EnumTab.KW, "while", 0, 0);
        tabelaSimbolos[palavra] = new InfIdentificador();

        palavra = new Token(EnumTab.KW, "write", 0, 0);
        tabelaSimbolos[palavra] = new InfIdentificador();

        palavra = new Token(EnumTab.KW, "read", 0, 0);
        tabelaSimbolos[palavra] = new InfIdentificador();

        palavra = new Token(EnumTab.KW, "num", 0, 0);
        tabelaSimbolos[palavra] = new InfIdentificador();
```

Classe/Diretório responsável por especificar as palavras chaves

```
//Insere o identificaor
public void insereIdentificador(Token palavra, InfIdentificador ident)
{
    tabelaSimbolos.Add(palavra, ident);
}
```

Método que insere o identificador

```
// Retorna um identificador de um determinado token
public InfIdentificador obtemIdentificador(Token palavra)
{
    InfIdentificador infoIdentificador = tabelaSimbolos[palavra];
    return infoIdentificador;
}
```

Método que obtém o identificador

```
//Pesquisa na tabela de símbolos se há algum token com determinado
  lexema Esse método diferencia ID e KW
public Token retornaToken(string lexema, int linha, int coluna)
{
    foreach (Token token in tabelaSimbolos.Keys)
    {
        if (token.getLexema().Equals(lexema))
        {
            token.nLinha = linha;
            token.nColuna = coluna;
            return token;
        }
    }
    return null;
}
```

Pesquisa na tabela de símbolos se existe algum token de acordo com o que foi passado por parâmetro e retorna caso exista.

```
//Saída
public override string ToString()
{
    string mensagemSaida = " ";
    int posicao = 1;

    foreach (Token token in tabelaSimbolos.Keys)
    {
        mensagemSaida += (("Posição: " + posicao + ": \t " +
          token.ToString()) + "\n");
    }

    return mensagemSaida;
}
```

Método responsável por retornar a saída caso verdadeira

```
public class InfIdentificador
{
    public string tipo;
}
```

Método que declara uma variável do tipo string, sua funcionalidade é usada no momento de encontrar um identificador.

```
public enum EnumTab
{
    EOF,

    //Operadores
    OP_ASS,
    OP_EQ,
    OP_GT,
    OP_GE,
    OP_LT,
    OP_LE,
    OP_NE,
    OP_AD,
    OP_MIN,
    OP_MUL,
    OP_DIV,

    //Símbolos
    SMB_OBC,
    SMB_CBC,
    SMB_OPA,
    SMB_CPA,
    SMB_COM,
    SMB_SEM,

    //Palavras reservadas
    KW,

    //Identificador
    ID,
```

Classe identificadora dos operadores/símbolos/identificador/constantes

OBS: Arquivos completos do código fonte encontra-se em anexo (disponível em PDF e cs)

Conclusão

O desenvolvimento dessa primeira parte foi de grande criatividade, raciocínio e pesquisa. A linguagem C# é vasta e complexa e requer uma constante pesquisa de métodos/funções e bibliotecas para o desenvolvimento do projeto. Foram estudados materiais desde tipos de variáveis a funções de análise de linhas com caracteres específicos.

A linguagem C# atendeu as expectativas necessárias para o desenvolvimento mesmo que o exemplo foi na linguagem Java.

Ultrapassando as dificuldades de bibliotecas/métodos foi possível aperfeiçoar o trabalho e o código, deixando-o de certa forma eficiente e atendendo os requisitos necessários.

Referências bibliográficas

1. https://pt.stackoverflow.com/questions/75848/gravar-e-recuperar-informa%C3%A7%C3%B5es-em-arquivos
2. http://www.vbmania.com.br/index.php?modulo=forum&metodo=abrir&id=443498
3. https://social.msdn.microsoft.com/Forums/pt-BR/73b2405f-6dd7-4b84-ad40-f4b9e5bfdb2b/identificao-de-quebra-de-linha-em-c?forum=webgeralpt
4. https://msdn.microsoft.com/pt-br/library/system.console.readline(v=vs.110).aspx
5. https://stackoverflow.com/questions/9210428/how-to-convert-class-into-dictionarystring-string/9210493
6. https://msdn.microsoft.com/pt-br/library/b4sc8ca8(v=vs.110).aspx
7. https://msdn.microsoft.com/pt-br/library/system.char.isletterordigit(v=vs.110).aspx
8. https://msdn.microsoft.com/en-us/library/system.net.http.httpcontentextensions.readasasync(v=vs.118).aspx
9. https://msdn.microsoft.com/pt-br/library/system.io.filestream.read(v=vs.110).aspx
10. https://msdn.microsoft.com/en-us/library/a63811ah(v=vs.110).aspx
11. https://stackoverflow.com/questions/12365346/how-to-assign-a-dictionary-value-to-a-session-in-asp-net
12. https://stackoverflow.com/questions/32742729/get-type-of-enum-in-net-class
13. https://social.msdn.microsoft.com/Forums/vstudio/pt-BR/5b7418cd-1473-42d2-89bb-6f8c14d21270/c-list-how-to-read-the-data?forum=csharpgeneral
14. https://imasters.com.br/linguagens/c-sharp/c-lendo-e-escrevendo-em-arquivos-textos-streamreaderstreamwriter/?trace=1519021197&source=single
15. http://pt.wikipedia.org/wiki/C_Sharp 2.
16. http://msdn.microsoft.com/en-us/library/system.text.regularexpressions.regex.split.aspx 4.
17. http://msdn.microsoft.com/en-us/library/aa332139(VS.71).aspx 5.
18. http://www.devarticles.com/c/a/C-Sharp/Introduction-to-Objects-and-Classes-in-C-Sharp-Part-2

unibh

GITHUB

DESENVOLVIMENTO DO TRABALHO

https://github.com/stellaoliveirabertt/compilador