Documentação - Analisador Léxico

01/04/2018

Greg A. Sullivan, Samuel H. Rizzon
UNIBH

Visão geral

Este documento visa especificar detalhadamente o código do projeto de um analisador léxico para a matéria de compiladores.

Informações sobre o código

Linguagem utilizada: C#

Todo o código foi escrito em C#. O critério para escolha da linguagem foi o conhecimento dos desenvolvedores.

IDE utilizada: Visual Studio

Visual Studio é a melhor IDE, sem dúvidas.

Instalação e utilização do código

Para executar o código, é necessária a instalação do Visual Studio, juntamente com a linguagem C#. Foi utilizado também o Windows Forms, em alguns casos é necessária a instalação do mesmo através de pacotes NuGet.

Estrutura dos arquivos

Lexer >

- Analise_Lexica.cs
- Tag.cs
- Token.cs
- TS.cs

Todo o analisador está dentro da pasta Lexer. Segue abaixo a explicação de cada arquivo detalhadamente.

Analise_Lexica.cs

Este é o arquivo principal, nele contém basicamente todo o processo de análise léxica, juntamente com o tratamento de erros.

Construtor

Analise_Lexica(String caminho, TS TabelaSimbolos)

Descrição: Construtor da classe.

Parâmetros:

- **caminho**: string contendo o caminho do arquivo que irá ser analisado.
- **TabelaSimbolos**: objeto da classe TS, onde contém todos os símbolos.

Métodos

Abrir_Arquivo (String caminho)

Descrição: Este método tem como objetivo abrir o arquivo para ser analisado. Caso dê algum erro na abertura do arquivo, um erro é retornado.

Parâmetros:

caminho: string contendo o caminho do arquivo que irá ser analisado.

Fechar_Arquivo ()

Descrição: Este método tem como objetivo fechar o arquivo que estava aberto para a análise.

GetErro (String Mensagem, ref RichTextBox textError)

Descrição: Este método tem como objetivo escrever no arquivo de texto os erros retornados durante a análise.

Parâmetros:

mensagem: string contendo a mensagem de erro.

textError: objeto do tipo RichTextBox que irá receber a mensagem de erro.

RetornaPonteiro ()

Descrição: Este método tem como objetivo retornar o ponteiro do analisador.

Funções

Boolean FimArquivo ()

Descrição: Este método tem como objetivo retornar se a leitura do arquivo chegou ao fim. Ele retorna um valor do tipo Boolean.

true: caso o arquivo tenha chegado ao fim.

false: caso o arquivo não tenha chegado ao fim.

Token ProximoToken (ref RichTextBox textError)

Descrição: Este método tem como objetivo encontrar e retornar o token encontrado ao ler o arquivo.

Parâmetros:

textError: objeto do tipo RichTextBox que irá receber a mensagem de erro.

Tag.cs

Este arquivo contém um Enum de todas as TAGs do analisador léxico. Essas tags servem para saber qual o tipo de token recebido.

Construtor

Não existe um construtor, por se tratar de um Enum.

Tipos de tags

As tags podem ser agrupadas em tipos. Seguem os tipos de tags utilizadas.

FIM ARQUIVO

EOF.

OPERADORES

OP_EQ;

OP_NE;

OP_GT;

OP_LT;

OP_GE;

OP_LE;

OP_AD;

OP_MIN;

OP_MUL;

OP_DIV;

OP_ASS.

SÍMBOLOS

SMB_OBC;

SMB_CBC;

SMB_OPA;

SMB_CPA;

SMB_COM;

SMB_SEM.

PALAVRA RESERVADA

KW.

IDENTIFICADOR

ID.

LITERAL

LIT.

CONSTANTES

CON_NUM;

CON_CHAR.

Token.cs

Este arquivo contém o modelo de Token padronizado.

Construtor

Token (Tag classe, String lexema, int linha, int coluna, string error = "")

Descrição: Construtor da classe.

Parâmetros:

classe: tipo do token que foi definido no Enum de tags.

lexema: string contendo o lexema encontrado ao analisar o arquivo.

linha: linha onde se encontra o token.

coluna: coluna onde se encontra o token.

error: string com a mensagem de erro, no caso do encontro de erros durante a análise.

Funções

String toString()

Descrição: Este método tem como objetivo retornar o lexema em formato de string e com a formatação de retorno de token definida pelo professor.

Exemplo de retorno: <KW, "program">

TS.cs

Este arquivo contém a tabela de símbolos.

Construtor

TS ()

Descrição: Construtor da classe.

Funções

Token RetornaToken(String lexema)

Descrição: Este método tem como objetivo verificar se o token lido pelo analisador existe na tabela de símbolos, e retornar o próprio token. Caso não exista, ele irá adicionar um novo símbolo na tabela.

Exemplo de retorno: Objeto do tipo Token.

String toString()

Descrição: Este método tem como objetivo retornar em uma string toda a tabela de símbolos.

Exemplo de retorno: Tabela de simbolos:

posicao 1: <KW, "program">

[...]

posicao N: <TIPO, “LEXEMA”>

Utilização na prática

Para podermos utilizar o analisador léxico, na linguagem C#, é necessário algumas linhas de código para a utilização das classes. Iremos mostrar e explicar como deve ser feito a chamada do analisador.

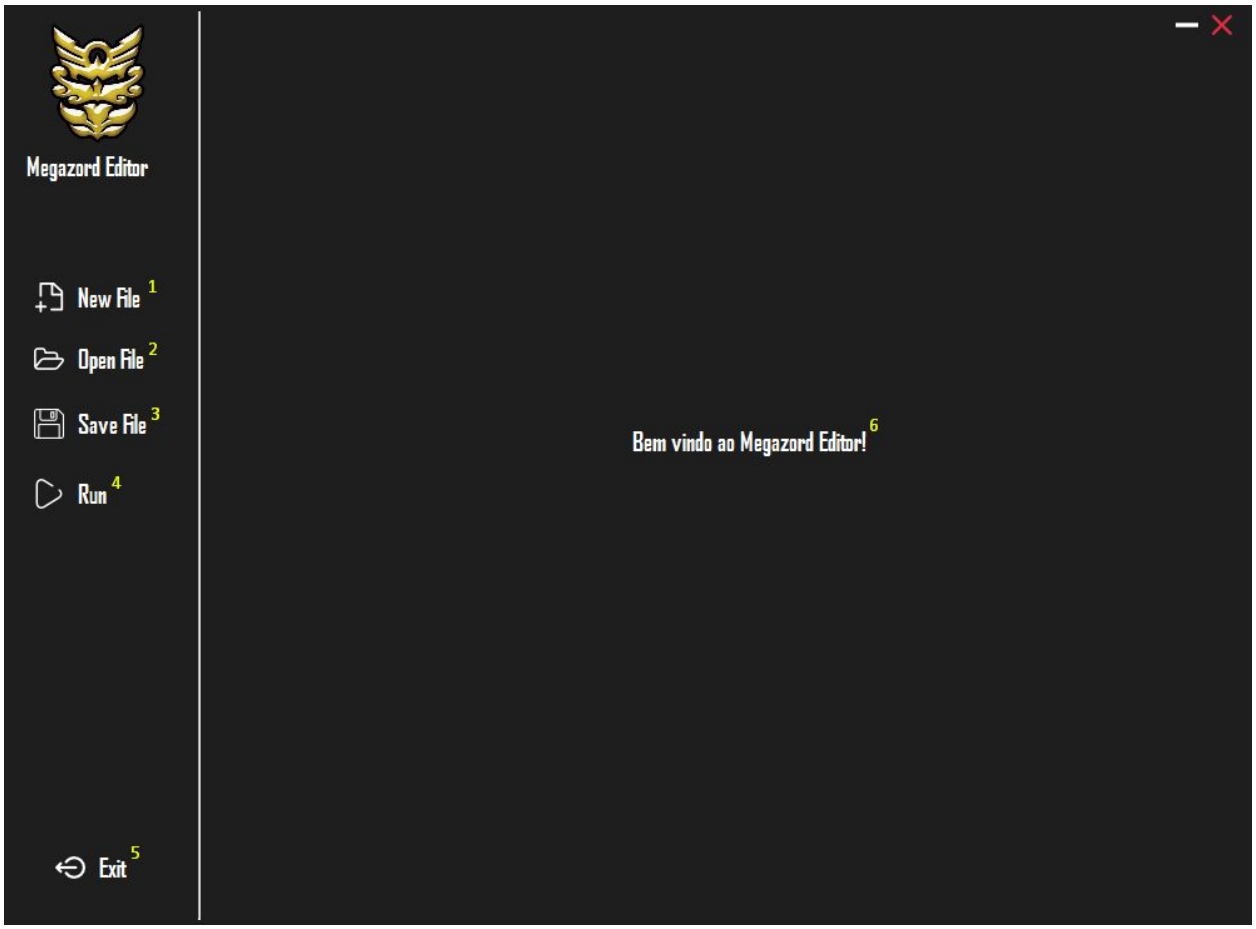
```
private void RunFile()
{
	TS TabelaSimbolos = new TS();
	Token token;
	try
	{
		/* Inicializa a classe principal do analisador, enviando o caminho onde
está o arquivo, e a tabela de		símbolos, que já foi iniciada anteriormente
*/
		Analise_Lexica analise = new Analise_Lexica(fileOpenNow, TabelaSimbolos);
		do
		{
			//Chama a classe para verificar o próximo token
			token = analise.ProximoToken(ref textOutput, ref textError);

			if (token != null)
			{
				//Imprimir o token
			}
		} while (token != null && token.Classe != Lexer.Tag.EOF);
	}
	catch (Exception)
	{
		throw;
	}
}
```

Observação: este código faz parte da classe TextEditor.cs, que é referente ao editor de texto criado pelos autores do analisador para uma melhor interação com o usuário.

Megazord Editor

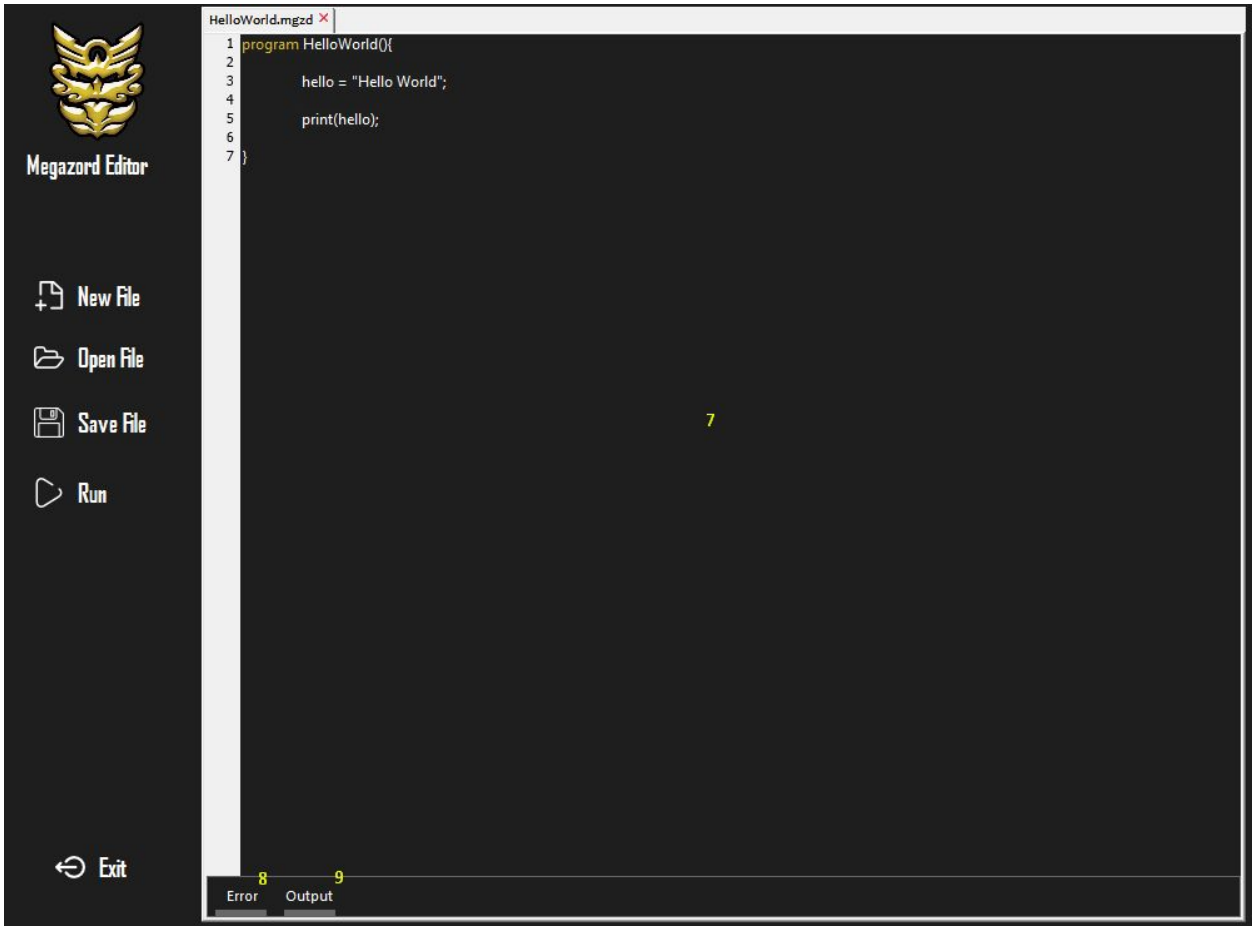
O *Megazord Editor* é um editor de texto criado no projeto do analisador, para que o usuário possa criar e buildar seus próprios códigos dentro do próprio editor.
A interface é bem simples. Segue abaixo uma explicação sobre a interface do editor:



Legenda: exemplo da tela inicial do *Megazod Editor*.

Informações:

1. Criar um novo arquivo
2. Abrir um arquivo existente. Os arquivo válidos são do tipo **.txt** e **.mgzd**.
3. Salvar o arquivo que está aberto.
4. Executar o analisador léxico.
5. Sair do programa.
6. Tela de boas vindas ao editor.



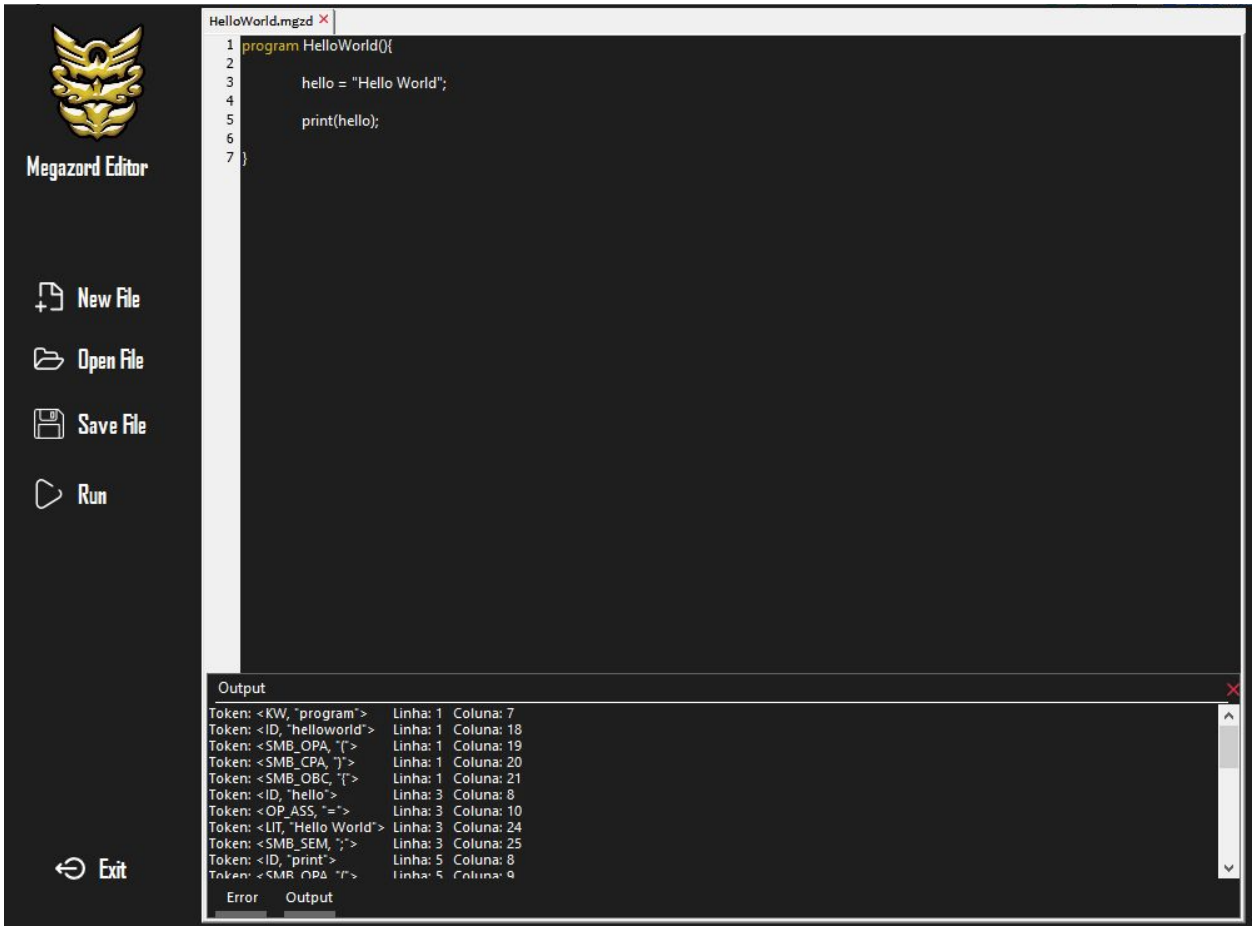
Legenda: exemplo da tela de edição do código.

Informações:

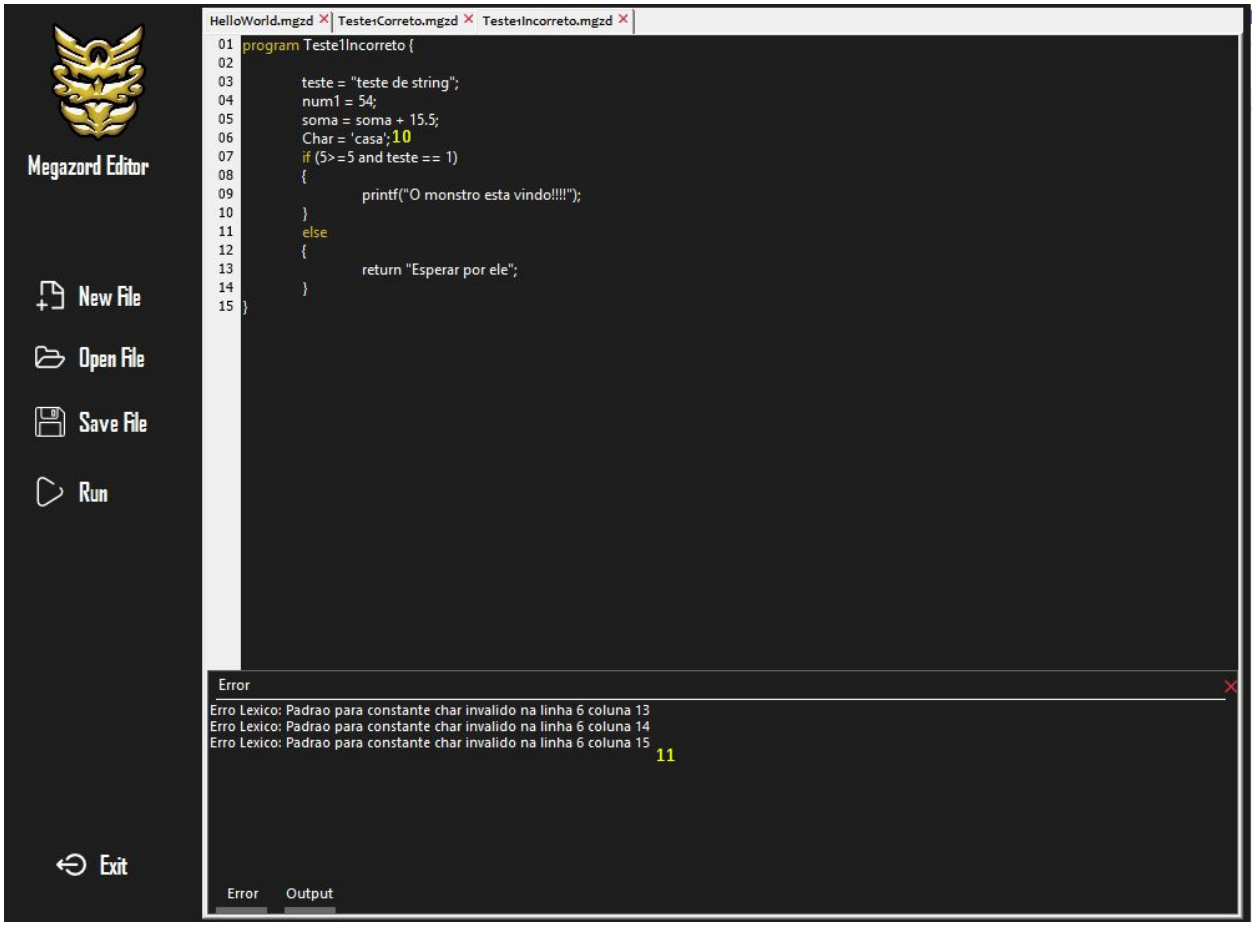
7. Tela de edição do código que será analisado pelo compilador.

8. Botão para visualização dos erros retornados pelo analisador, após a execução do arquivo.

9. Botão para a visualização do retorno do analisador, nesta aba serão mostrados todos os tokens, os erros, e a tabela de símbolos.



Legenda: exemplo da tela de output, após uma execução.



Legenda: exemplo da tela de erros, após uma execução.

Informações:

10. Existe um erro na linha 06, pois um char não pode conter mais de um caractere.

11. O analisador retorna o erro na aba de erros, informando o motivo do erro, e sua respectiva linha e coluna.