

Projeto de Compilador E1 de Análise Léxica

Prof. Lucas Mello Schnorr
schnorr@inf.ufrgs.br

1 Introdução

A primeira etapa consiste em fazer um analisador léxico utilizando a ferramenta de geração de reconhecedores `flex`. Tu deves manter o arquivo `tokens.h` (fornecido) sem modificações. A função principal deve estar em um arquivo `main.c` separado do arquivo `scanner.l` para facilitar testes automáticos que utilizam uma função principal parecida com a fornecida em anexo.

2 Funcionalidades Necessárias

2.1 Definir expressões regulares

Reconhecimento dos lexemas correspondentes aos tokens descritos na seção **Descrição dos Tokens** abaixo, unicamente através da definição de expressões regulares no arquivo da ferramenta `flex`. Cada expressão regular deve estar associada a pelo menos um tipo de token. Classificar os lexemas reconhecidos em tokens retornando as constantes definidas no arquivo `tokens.h` fornecido ou códigos ASCII para caracteres simples.

2.2 Contagem de linhas

Controlar o número de linha do arquivo de entrada. Uma função cujo protótipo é `int get_line_number(void)` deve ser implementada e deve retornar o número da linha atual no processo de reconhecimento de tokens. Ela é utilizada nos testes automáticos. Lembre-se que a primeira linha de qualquer arquivo dado como é entrada é a linha número um.

2.3 Ignorar comentários

Ignorar comentários no formato C99: tudo o que segue a partir de `//` e tudo que está compreendido entre `/*` e `*/`. As linhas devem ser contabilizadas mesmo dentro de comentários do segundo tipo. Espaços devem ser igualmente ignorados.

2.4 Lançar erros léxicos

Lançar erros léxicos ao encontrar caracteres inválidos na entrada, retornando o token de erro `TK_ERRO`.

3 Descrição dos Tokens

Existem tokens que correspondem a caracteres, como vírgula, ponto-e-vírgula, parênteses. Para estes, é mais conveniente usar seu próprio código ASCII, convertido para inteiro, como valor de retorno que os identifica. Para os demais tokens, como palavras reservadas e identificadores, utiliza-se uma constante com valores superiores ao maior valor da tabela ASCII, definida em `tokens.h`. Os tokens se enquadram em diferentes categorias descritas a seguir.

3.1 Palavras Reservadas da Linguagem

As palavras reservadas (PR) da linguagem são as seguintes, acompanhadas dos tokens correspondentes no arquivo `tokens.h`:

PR	Token
<code>int</code>	<code>TK_PR_INT</code>
<code>float</code>	<code>TK_PR_FLOAT</code>
<code>bool</code>	<code>TK_PR_BOOL</code>
<code>char</code>	<code>TK_PR_CHAR</code>
<code>if</code>	<code>TK_PR_IF</code>
<code>then</code>	<code>TK_PR_THEN</code>
<code>else</code>	<code>TK_PR_ELSE</code>
<code>while</code>	<code>TK_PR_WHILE</code>
<code>input</code>	<code>TK_PR_INPUT</code>
<code>output</code>	<code>TK_PR_OUTPUT</code>
<code>return</code>	<code>TK_PR_RETURN</code>
<code>for</code>	<code>TK_PR_FOR</code>

3.2 Caracteres Especiais

Os caracteres simples especiais empregados pela linguagem são listados abaixo separados apenas por espaços, e devem ser retornados com o próprio código ASCII convertido para inteiro. São eles:

`, ; : () { } + - * / % ^ < > = !`

3.3 Operadores Compostos

A linguagem possui operadores compostos (OC), além dos operadores representados por alguns dos caracteres da seção anterior. Os operadores compostos são:

OC	Token
<code><=</code>	<code>TK_OC_LE</code>
<code>>=</code>	<code>TK_OC_GE</code>
<code>==</code>	<code>TK_OC_EQ</code>
<code>!=</code>	<code>TK_OC_NE</code>
<code>&&</code>	<code>TK_OC_AND</code>
<code> </code>	<code>TK_OC_OR</code>

3.4 Identificadores

Os identificadores da linguagem são formados por uma sequência de pelo menos um caractere alfabético (letras maiúsculas ou minúsculas). Ao reconhecer um identificador, retornamos `TK_IDENTIFICADOR`.

3.5 Literais

Literais são formas de descrever constantes no código fonte.

- `TK_LIT_INT`: literais deste tipo são representados como repetições de um ou mais dígitos precedidos opcionalmente pelo sinal de negativo ou positivo.
- `TK_LIT_FLOAT`: literais deste tipo são formados como um inteiro seguido de ponto decimal e uma sequência de dígitos. A notação científica é possível para números ponto flutuantes utilizando um `E` ou `e` seguido de um número positivo ou negativo inteiro.
- `TK_LIT_FALSE`: o literal é o lexema `false`.
- `TK_LIT_TRUE`: o literal é o lexema `true`.
- `TK_LIT_CHAR`: literais deste tipo são representados por um único caractere entre aspas simples como por exemplo: `'a', ' ', '+'`.