Projeto de Compilador
Etapa 1 : Análise Léxica

Stéfano Drimon Kurz Mór e Gabriel Bronzatti Moro

8 de Agosto de 2016

O trabalho consiste no projeto e implementação de um compilador funcional para uma determinada gramática de linguagem de programação. A primeira etapa do trabalho consiste em fazer um analisador léxico utilizando a ferramenta de geração de reconhecedores `flex` e preencher uma tabela com os símbolos relevantes encontrados, incluindo em qual a linha do arquivo o lexema correspondente aparece.

1 Funcionalidades Necessárias

1.1 Definir Expressões Regulares

Reconhecimento dos lexemas correspondentes aos tokens descritos na Seção 2, 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. Os tipos de tokens estão listados no arquivo de configuração do `bison`.

1.2 Contagem de Linhas

Controlar o número de linha do arquivo de entrada. A função cujo protótipo é `comp_get_line_number()` (em `cc_misc.c`) deve ser implementada. Ela é utilizada nos testes automáticos. Arquivos começam pela linha número um.

1.3 Preencher a Tabela de Símbolos

A tabela de símbolos deve ser preenchida com os tokens:

- identificadores,
- literais (inteiros, flutuantes, caracteres, cadeia de caracteres).

Outros tokens devem estar ausentes da tabela de símbolos. A chave de cada entrada na tabela deve ser o **lexema** do token encontrado. O conteúdo de cada entrada na tabela de símbolos deve ser o número da linha onde o último lexema correspondente foi encontrado. Na ocorrência de múltiplos lexemas idênticos na entrada, somente o número da linha da última ocorrência deve estar registrado na entrada correspondente.

1.4 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.

1.5 Lançar Erros Léxicos

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

1.6 Listar o Conteúdo da Tabela de Símbolos

Implementar a função `comp_print_table`, em `cc_misc.c` de forma a listar todas as entradas da tabela de símbolos. Deve-se utilizar obrigatoriamente a função `void cc_dict_etapa_1_print_entrada (char *key, int line)` para imprimir uma entrada. Esta função será utilizada na avaliação automática para averiguar se a solução insere somente os tokens que devem ser inseridos na tabela de símbolos.

2 Descrição dos Tokens

Existem tokens que correspondem a caracteres particulares, como vírgula, ponto-e-vírgula, parênteses, para os quais é mais conveniente usar seu próprio código `ASCII`, convertido para inteiro, como valor de retorno que os identifica. Para os tokens compostos, como palavras reservadas e identificadores, utiliza-se uma constante, com recursos do `bison`, com um código maior do que 255 para representá-los. Os tokens se enquadram em diferentes categorias:

1. palavras reservadas da linguagem,
2. caracteres especiais,
3. operadores compostos,
4. identificadores e
5. literais.

O analisador léxico deve, para as categorias de palavras reservadas, operadores compostos, identificadores e literais, retornar o token correspondente de acordo o que está definido no arquivo `parser.y` (veja as linhas que começam por `%token`). Para a categoria de caracteres especiais, o analisador léxico deve retornar o código `ASCII` através de uma única regra que retorna `yytext[0]`.

2.1 Palavras Reservadas da Linguagem

As palavras reservadas da linguagem são:

```
int float bool char string if then else
while do input output return const static
foreach for switch case break continue class
private public protected
```

2.2 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:

```
,   ;   :   (   )   [   ]   {   }   +   -
*   /   <   >   =   !   &   $   %   #   ^
```

2.3 Operadores Compostos

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

```
<=    >=    ==    !=
&&    ||    >>    <<
```

2.4 Identificadores

Os identificadores da linguagem são formados por um caractere alfabético seguido de zero ou mais caracteres alfanuméricos, onde considera-se caractere alfabético como letras maiúsculas ou minúsculas ou o caractere sublinhado e onde dígitos são `0`, `1`, `2`, ..., `9`.

2.5 Literais

Literais são formas de descrever constantes no código fonte. Literais do tipo `int` são representados como repetições de um ou mais dígitos precedidos opcionalmente pelo sinal de negativo ou positivo. Literais em `float` são formados como um inteiro seguido de ponto decimal e uma sequência de dígitos. Literais do tipo `bool` podem ser `false` ou `true`. Literais do tipo `char` são representados por um único caractere entre entre aspas simples como por exemplo o `'a'`, `'='` e `'+'`. Literais do tipo `string` são qualquer sequência de caracteres entre aspas duplas, como por exemplo `"meu nome"` ou `"x = 3;"`. Os literais do tipo `char` e `string` devem ser inseridos na tabela de símbolos sem as aspas que os identificam no código fonte.

3 Regras Gerais

Veja no Moodle as regras gerais de entrega.