UFRGS - Informática - INF01147 - Compiladores - 2021/2
Turma B - Prof. Marcelo Johann

Trabalho Prático - Especificação da Etapa 1:
Análise Léxica e Inicialização de Tabela de Símbolos

Resumo:

O trabalho consiste na implementação de um compilador funcional. Esta primeira etapa do trabalho consiste em fazer um analisador léxico utilizando a ferramenta de geração de reconhecedores *lex* (ou *flex*) e inicializar uma tabela global de símbolos encontrados.

Funcionalidades necessárias:

A sua análise léxica deve fazer as seguintes tarefas:

a. reconhecer as expressões regulares que descrevem cada tipo de lexema;
b. classificar os lexemas reconhecidos em *tokens* retornando as constantes definidas no arquivo `tokens.h` fornecido ou códigos *ascii* para caracteres simples;
c. incluir os identificadores e os literais (inteiros, caracteres e *strings*) em uma tabela de símbolos global implementada com estrutura *hash*;
d. controlar o número de linha do arquivo fonte, e fornecer uma função declarada como `int getLineNumber(void)` a ser usada nos testes e pela futura análise sintática;
e. ignorar comentários de única linha e múltiplas linhas;
f. informar erro léxico ao encontrar caracteres inválidos na entrada, retornando o *token* de erro;
g. definir e atualizar uma variável global e uma função `int isRunning(void)`, que mantém e retorna valor *true* (diferente de 0) durante a análise e muda para *false* (igual a 0) ao encontrar a marca de fim de arquivo;

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, cria-se uma constante (`#define` em C ANSI) com um código maior do que 255 para representá-los.

Os *tokens* representam algumas categorias diferentes, como palavras reservadas, operadores de mais de um caractere e literais, e as constantes definidas no código do trabalho são precedidas por um prefixo para melhor identificar sua função, separando-as de outras constantes que serão usadas no compilador.

Palavras reservadas

As palavras reservadas da linguagem neste semestre são: `char`, `int`, `float`, `if`, `then`, `else`, `while`, `goto`, `read`, `print`, `return`. Para cada uma deve ser retornado o *token* correspondente.

Caracteres especiais

Os caracteres simples especiais empregados pela linguagem são listados abaixo (estão separados apenas por espaços), e devem ser retornados com o próprio código *ascii* convertido para inteiro. Você pode fazer isso em uma única regra léxica que retorna `yytext[0]`. São eles:

```
, ; : ( ) [ ] { } + - * / < > =
```

Operadores Compostos

A linguagem possui, além dos operadores representados por alguns dos caracteres acima, operadores compostos, que necessitam mais de um caractere (somente dois) para serem representados no código fonte. São somente quatro operadores relacionais, conforme a tabela:

Source Representation	Returned Token
<=	OPERATOR_LE
>=	OPERATOR_GE
==	OPERATOR_EQ
!=	OPERATOR_DIF

Identificadores

Os identificadores da linguagem são usados para designar variáveis, vetores e nomes de funções, são formados por uma sequência de um ou mais caracteres alfabéticos minúsculos, isto é, apenas no intervalo [a-z], e também os caracteres *underline* (´_´) e ´-´, ou seja, o mesmo símbolo do operador de subtração, que neste caso não pode aparecer sozinho.

Literais

Literais são formas de descrever constantes no código fonte. Literais inteiros são formados por uma sequência de um ou mais dígitos decimais. Literais do tipo caracter são representados por um único caractere entre *aspas simples* (mais precisamente apóstrofo, ASCII decimal 39), como por exemplo: `'a'`, `'X'`, `'-'`. Não existem literais específicos para o tipo de dado float. Literais do tipo *string* são qualquer sequência de caracteres entre aspas duplas, como por exemplo `"meu nome"` ou `"Mensagem!"`, e servem apenas para imprimir mensagens com o comando ´`print`´. *Strings* consecutivas não podem ser consideradas como apenas uma, o que significa que o caractere de aspas duplas não pode fazer parte de uma *string*. Para incluir os caracteres de aspas duplas e final de linha, devem ser usadas sequências de escape, como "\"" e "\n".

Comentários

Comentários de uma única linha começam em qualquer ponto com a sequência "\\" e terminam na próxima marca de final de linha, representada pelo caractere '\n'. Comentários de múltiplas linhas iniciam pela sequência "*" e terminam pela sequência "*\", sendo que podem conter quaisquer caracteres, que serão todos ignorados, incluindo uma ou mais quebras de linha, as quais, entretanto, devem ser contabilizadas para controle do número de linha.

Controle e organização do seu código fonte

Você deve manter o arquivo `tokens.h` intacto, e separar a sua função main em um arquivo especial chamado `main.c`, já que a função main não pode estar contida no código de `scanner.l`. Isso é necessário para facilitar a automação dos testes, que utilizará uma função `main` especial escrita pelo professor, substituindo a que você escreveu para teste e desenvolvimento. Você deve usar essa estrutura de organização, manter os nomes `tokens.h` e `scanner.l`. Instruções mais detalhadas sobre o formato de submissão do trabalho e cuidados com detalhes específicos estão em outro documento separado.

Atualizações e Dicas

Verifique regularmente os documentos e mensagens da disciplina para informar-se de alguma eventual atualização que se faça necessária ou dicas sobre estratégias que o ajudem a resolver problemas particulares. Em caso de dúvida, consulte o professor.

Porto Alegre, 21 de Janeiro de 2022
Atualizado em 25/01/2022

Anexo - Código tokens.h

```
/*
 * Lista dos tokens, com valores constantes associados.
 * Este arquivo serah posterioremente substituido, nao acrescente nada.
 * Os valores das constantes sao arbitrarios, mas nao podem ser alterados.
 * Cada valor deve ser distinto e fora da escala ascii.
 * Assim, nao conflitam entre si e com os tokens representados pelo proprio
 * valor ascii de caracteres isolados.
 */


#define KW_CHAR         256
#define KW_INT          257
#define KW_FLOAT        258

#define KW_IF           261
#define KW_THEN         262
#define KW_ELSE         263
#define KW_WHILE        264
#define KW_GOTO         265
#define KW_READ         266
#define KW_PRINT        267
#define KW_RETURN       268

#define OPERATOR_LE     270
#define OPERATOR_GE     271
#define OPERATOR_EQ     272
#define OPERATOR_DIF    273

#define TK_IDENTIFIER   280

#define LIT_INTEGER     281
#define LIT_CHAR        285
#define LIT_STRING      286

#define TOKEN_ERROR     290

/* END OF FILE */
```