Especificação do Projeto – Etapa 1

Análise Léxica e Inicialização da Tabela de Símbolos

Compiladores

Prof. Filipe de Aguiar Geissler

Data Limite para apresentação

Aula de revisão mais entrega no Classroom. Deverá ser demonstrado para toda a turma. Neste contexto preparem slides mais demonstração.

Descrição

O objetivo do projeto de compiladores é implementar um compilador funcional. A etada 1 do projeto de compiladores consiste na implementação do analisador léxico com uso da ferramenta ***lex*** (ou ***flex***) juntamente com a inicialização de uma tabela de símbolos.

Requisitos de Implementação

A análise léxica, como visto em aula, tem as seguintes atribuições:

1. Realizar o reconhecimento dos lexemas da linguagem por meio de expressões regulares;
2. Classificar os lexemas em tokens retornando o código especificado no header tokens.h fornecido no Moodle ou código ASCII para caracteres simples;
3. Incluir os caracteres ou literais (inteiros, caracteres e strings) em uma tabela de símbolos com estrutura de dados tipo ***hash***;
4. Controlar o número de linhas do código fonte fornecendo uma função de acesso ao valor denominada como segue:

   int getLineNumber(void)

   Tal função será utilizada pelo analisador sintático em etapa posterior do projeto;
5. Ignorar comentários de única linha (//) ou múltiplas linhas (/* … */);
6. Informar erros léxicos ao encontrar caracteres inválidos no arquivo fonte retornando um código definido no header **token.h**;

7. Criar uma variável global de controle para manter o estado de operação do analisador sintático a qual será acessada por uma função como segue:

   int isScannerRunning(void)

   A mesma irá retornar 1 durante a análise léxica e 0 quando encontrar o final do arquivo fonte.

Linguagem Analisada

Os caracteres particulares da linguagem especificada neste documento, tais como vírgula, parênteses, entre outros, tem seu código de token sendo o próprio valor da tabela ASCII. Para os tokens compostos com palavras reservadas e identificadores utilizaremos uma constante definida com código maior que 255. A definição destes códigos é realizada com um #define em C ANSI. O arquivo tokens.h contém tais definições.

Palavras Reservadas

Para cada palavra reservada deve ser retornado um token (veja arquivo tokens.h) correspondente.

- int
- real
- bool
- char
- if
- else
- loop
- input
- output
- return

Caracteres Especiais

Um caracter especial retorna um token com valor correspondente ao da tabela ASCII. Para tal, é possível retornar o valor inteiro contido na variável yytext[0].

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

Operadores Compostos

Operadores compostos utilizam-se da composição de alguns caracteres especiais. A tabela abaixo apresenta 4 operadores relacionais e dois lógicos. Na tabela é apresentado o nome do token correspondente. O valor deste é definido em token.h.

operador	token
<=	OP_LE
>=	OP_GE
==	OP_EQ
!=	OP_NE
&&	OP_AND
||	OP_OR

Identificadores

Os identificadores da linguagem server para definir variáveis, vetores e nomes de funções ou procedimentos. Identificadores não podem ser iniciados por underline (_). Dígitos não são permitidos na composição de nomes da linguagem.

Literais

Literais são formar de descrever constantes no código fonte, como segue:

- inteiros: repetições de dígitos
- caracteres: representados por um único caractere entre aspas simples: 'a'
- string: sequências de caracteres entre aspas duplas: “Mensagem Teste”, servindo apenas para imprimir mensagens com o comando output. Para incluir o caractere de aspas duplas deve ser utilizada a contra barra como “\””. Os literais do tipo caractere e string podem ser inseridos na tabela de símbolos sem as aspas. Porém devem ser diferenciados pois tratam-se de tipos de literias diferentes;
- booleanos: inicialização de variáveis podem ser realizadas com as palavras TRUE ou FALSE em maiúsculo.

Organização do Código Fonte do Projeto

- As ferramentas make, lex ou flex e gcc devem ser utilizadas;

- O arquivo **tokens.h** não pode ser alterado;
- O arquivo com as expressões regulares deve ser chamado de **scanner.l**;
- A função **main** deve ser mantida em um arquivo especial chamado **main.c** com chamadas de teste **yylex** na função main(). Nenhuma outra implementação deve constar neste arquivo ou #include de arquivos fonte, visto que tal arquivo será substituído pelo arquivo do professor para realização dos testes automatizados;
- Deverá ser implementada uma função `void initMe(void)` de onde todo e qualquer código de inicialização necessário como inicialização da tabela *hash* deve ser chamada. Esta função será chamada pela função main testadora do professor no processo de testes automatizados;
- Deve ser possível compilar o projeto com o uso de um **Makefile** (consulte exemplos disponibilizados no Moodle) através do comando **make**.

Formato de Entrega

Cada dupla deve compactar o conteúdo da pasta de trabalho com o comando

```
$ tar cvzf etapa1.tgz
```

dentro do mesmo diretório. Não utilizar outros programas, formatos, comandos, nomes ou organizações de diretórios para esta operação.
O arquivo etapa1.tgz gerado deve ser submetido no Classroom.
No diretório de trabalho deve existir o arquivo Makefile com todos os comandos necessários à compilação do programa. Ao executar o comando make deverá ser gerado o executável no mesmo diretório denominado de "etapa1".

O trabalho será apresentado em sala de aula para todos os colegas uma aula antes da prova. Lembrando que a data da prova segue sempre o calendário acadêmico.

Instruções para realização do trabalho

- O trabalho deve ser desenvolvido por dois alunos não podendo ser realizado de forma individual;
- Deve ser informado no Classroom via comentário na tarefa **Etapa 1** o nome dos integrantes das duplas.