

## Especificação da Etapa 1 do Projeto de Compilador

### Análise Léxica e Inicialização da Tabela de Símbolos

O trabalho como um todo consiste no projeto e implementação de um compilador funcional para uma determinada gramática de linguagem de programação. Esta primeira etapa do trabalho consiste em fazer um analisador léxico utilizando a ferramenta de geração de reconhecedores **flex** e inicializar uma tabela de símbolos encontrados, incluindo atributos como qual a linha no arquivo de um lexema.

#### 1 Funcionalidades Necessárias

A sua análise léxica deve realizar as seguintes tarefas:

1. **Reconhecer as expressões regulares** que descrevem cada tipo de lexema, classificando-os em *tokens* e retornando as contantes definidas no arquivo `tokens.h` fornecido ou códigos *ASCII* para caracteres simples
2. **Preencher a tabela de símbolos** com os identificadores e todos os literais (inteiros, ponto flutuantes, caracteres e cadeia de caracteres) através do uso do dicionário implementado na etapa anterior. Todos os identificadores e literais devem ter pelo menos um atributo na tabela de símbolos que consiste no número da linha do arquivo onde ele foi encontrado.
3. **Controlar o número de linha** do arquivo fonte, implementando uma função que deve ser obrigatoriamente declarada como `int getLineNumber(void)` e será usada nos testes automáticos e pela futura análise sintática para o relatório de erros.
4. **Ignorar comentários** no formato C99 de única linha e múltiplas linhas
5. **Lançar erros léxicos** ao encontrar caracteres inválidos na entrada, retornando o *token* de erro

#### 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, cria-se uma constante – com o uso de `#define` – 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 são precedidas por um prefixo para melhor identificar a sua função, separando-as de outras constantes que serão usadas no compilador.

##### 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`. Para cada uma deve ser retornado um *token* correspondente de acordo com o arquivo `tokens.h`.

##### 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:

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

Dica: o código *ASCII* desses caracteres pode ser retornado construindo uma única regra que retorna `yytext[0]`.

##### 2.3 Operadores Compostos

A linguagem possui operadores compostos, além dos operadores representados por alguns dos caracteres da seção anterior. Estes operadores compostos necessitam de dois caracteres para serem representados no código fonte e devem ser retornados através do *token* correspondente de acordo com o arquivo `tokens.h`. Os operadores compostos são:

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

## 2.4 Identificadores

Os identificadores da linguagem são usados para designar nomes de variáveis e funções. Eles são formados por um caracter alfabético seguido de zero ou mais caracteres alfabéticos ou dígitos, onde considera-se caracter alfabético como letras maiúsculas ou minúsculas ou o caracter sublinhado `_` e onde são dígitos: 0, 1, 2, ..., 9. Eles devem ser inseridos na tabela de símbolos e retornados juntamente com o *token* correspondente (veja o arquivo *tokens.h*).

## 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. Literais em `float` são formados como um inteiro seguido de ponto decimal e uma sequência de dígitos decimais. Literais do tipo `bool` podem ser `false` ou `true`. Literais do tipo `char` são representados por um único caractere 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;"`, e servem apenas para imprimir mensagens com o comando `output`. Os literais do tipo `char` e `string` devem ser inseridos na tabela de símbolos sem as aspas que os identificam no código fonte.

## 3 Controle e Organização da Solução

O arquivo `tokens.h` deve permanecer intacto. A função `main` deve estar em um arquivo chamado `main.c`. Outros arquivos fontes são encorajados de forma a manter a modularidade do código fonte. A entrada para o *flex* deve estar em um arquivo com o nome `scanner.l`.

### 3.1 Git e Cmake

A solução desta etapa do projeto de compiladores deve ser feita sobre a etapa 0. Cada ação de commit deve vir com mensagens significativas explicando a mudança feita. Todos os membros do grupo devem ter feito ações de commit, pelo fato deste trabalho ser colaborativo. Estas duas ações – mensagens de commit e quem fez o commit – serão obtidas pelo professor através do comando `git log` na raiz do repositório solução do grupo. Os arquivos adicionais necessários para esta etapa podem ser obtidos através do seguinte comando (para atualizar o repositório já clonado na etapa 0):

```
$ git pull origin master
```

Note que o arquivo `scanner.l`, que deverá ser fortemente modificado para atender aos requisitos deste trabalho, está praticamente vazio. Novos arquivos de código fonte podem ser adicionados, modificando o arquivo `CMakeLists.txt`, para que ele seja incluído no processo de compilação do analisador léxico.

## 4 Atualizações e Dicas

Verifique regularmente o Moodle da disciplina e o final deste documento 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, não hesite em consultar o professor.