Compiladores

Aula 06: Geradores Léxicos e a Ferramenta Flex

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Março de 2026

Sumário

Relembrando: O Desafio da Implementação

Nas aulas anteriores, vimos que um Scanner é a simulação em software de um Autômato Finito Determinístico (AFD).

A Abordagem Manual (Ad-hoc):

- Consiste em criar um laço `while` gigante com `switch/case` e `if/else`.
- **Código Espaguete:** À medida que o vocabulário cresce, as condicionais tornam-se ilegíveis.
- **Erros de Lookahead:** Controlar os ponteiros para "espiar" o próximo caractere gera bugs sutis.
- **Manutenção Puniante:** Mudar a regra de um número exige reescrever a lógica de vários estados.

A Solução: Meta-programação

A Engenharia de Software nos ensina a abstrair tarefas repetitivas.

Se a conversão de uma **Expressão Regular** para um **AFD** é um processo puramente matemático (Thompson + Subconjuntos), por que não criar um programa que faça isso por nós?

O Papel dos Geradores Léxicos

Em vez de codificarmos as transições de estado, nós **declaramos as regras**. A ferramenta faz o trabalho pesado e gera o código C otimizado. É exatamente este o papel do **Flex**.

O que é o Flex?

O **Flex (Fast Lexical Analyzer Generator)** é um programa que escreve programas.

- **Entrada:** Um arquivo de especificação (`.l` ou `.lex`) contendo pares de Expressões Regulares e blocos de código em C.
- **Saída:** Um arquivo em C (tradicionalmente `lex.yy.c`) contendo a função de simulação do autômato (a função `yylex()`).

O Pipeline de Compilação

Não executamos o arquivo `.l` diretamente. O processo envolve duas etapas:

```
# Passo 1: O Flex processa as regras e gera o código C (lex.yy.c)
$ flex scanner.l

# Passo 2: O GCC compila o código gerado em um executável
# Nota: A flag -lfl anexa a biblioteca de execução do Flex
$ gcc lex.yy.c -lfl -o meu_scanner

# Passo 3: Executamos o scanner passando um código-fonte de teste
$ ./meu_scanner < codigo_fonte.pas
```

A Anatomia do Arquivo `.l`

A sintaxe do Flex é rigidamente dividida em **três seções**, separadas pelo delimitador %%. Essa estrutura permite mesclar definições regulares com código C nativo.

A Estrutura:

1. Definições
   %%
2. Regras
   %%
3. Código do Usuário

O que vai em cada lugar?

- **Definições:** `#include`, variáveis globais e macros de ERs.
- **Regras:** O mapeamento de Padrões para Ações C.
- **Código:** A função `main` e tratamentos de erro.

As Variáveis Mágicas do Flex

Na seção de Regras, quando um padrão é válido, o Flex preenche variáveis globais **antes** de executar o seu bloco de código em C:

`yytext`: Um ponteiro (`char*`) que contém a string exata (o lexema) que acabou de ser casada.

`yyleng`: O tamanho (em inteiro) da string contida em `yytext`.

Exemplo de Uso

```
[0-9]+  { printf("O número é: %d\n", atoi(yytext)); }
```

Juntando tudo: Mini-Pascal no Flex

```
%{
    #include <stdio.h>
    int num_linhas = 1;
%}
DIGITO      [0-9]
ID          [a-zA-Z_][a-zA-Z0-9_]*
%%
"program"   { printf("KW_PROGRAM\n"); }
"begin"     { printf("KW_BEGIN\n"); }
"end"       { printf("KW_END\n"); }
":="        { printf("OP_ASSIGN\n"); }
{ID}        { printf("ID: %s (Tam: %d)\n", yytext, yyleng); }
{DIGITO}+   { printf("NUM_INT: %d\n", atoi(yytext)); }
"{"[^}]*"}" { /* Ignora comentarios Pascal: ex: { teste } */ }
\n          { num_linhas++; }
[ \t]+      { /* Ignora espacos em branco */ }
.           { printf("Erro na linha %d: %s\n", num_linhas, yytext); }
%%
int main() {
    yylex(); // Inicia a varredura
    return 0;
}
```

O Problema da Ambiguidade Léxica

O que acontece se a fita de entrada contiver o seguinte código:

```
programName := 10;
```

Quando o scanner lê o `"p"`, `"r"` até o `"m"`, ele tem um casamento perfeito para a regra da palavra-chave `program`.

A Dúvida: O Flex deve retornar `KW_PROGRAM` imediatamente, ou continuar lendo a palavra inteira (`programName`) para retornar um Identificador (`ID`)?

Regra 1: Maior Casamento (Maximal Munch)

A Regra: O scanner sempre consome a *maior quantidade possível de caracteres* da entrada que formem um padrão válido.

- Padrão 1 (`"program"`) casaria **7** caracteres.
- Padrão 2 (`{ID}`) casaria **11** caracteres.

O Vencedor

O Padrão 2. O Flex consome a palavra inteira como um único `ID`. Isso garante que prefixos de variáveis não sejam confundidos com palavras reservadas!

Regra 2: Prioridade pela Ordem

E quando ocorre um empate no princípio do Maior Casamento? Exemplo: A fita contém exatamente a palavra solta `"begin"`.

- A regra `"begin"` lê 5 caracteres.
- A regra {`ID`} também lê 5 caracteres.

A Regra: Se houver empate no tamanho do lexema, vence a regra que foi **escrita primeiro** (mais acima) no arquivo `.l`.

A Regra de Ouro em Compiladores

Em ferramentas de parsing (Flex, JFlex, ANTLR), **sempre** declare as regras de Palavras Reservadas antes das regras genéricas de Identificadores.

Resumo da Aula

- Escrever *scanners* ad-hoc é custoso e inviável para linguagens ricas.
- O **Flex** abstrai a matemática dos autômatos, gerando matrizes de transição eficientes em C.
- Um arquivo `.l` possui Definições (código C + macros), Regras (ER $\rightarrow$ Ação) e Código Auxiliar.
- As variáveis `yytext` e `yyleng` servem de ponte entre a string casada e a semântica do compilador.
- Ambiguidades são resolvidas de forma estrita via *Maximal Munch* e prioridade de declaração.

Referências

- **AHO, A. V. et al.** *Compiladores: Princípios, Técnicas e Ferramentas*. 2ª Edição. Cap. 3.
- **LEVINE, J.** *flex & bison*. O'Reilly Media. (O guia de cabeceira para programação com Flex).
- **COOPER, K.; TORCZON, L.** *Engineering a Compiler*.

Dúvidas?

Obrigado!

Email: alessio@cefetmg.br
Web: alessiojr.com