

Análise léxica

O papel do analisador léxico

Prof. Edson Alves

Faculdade UnB Gama

Sumário

1. Gerador de analisadores léxicos

Flex

- ▶ Flex (*Fast Lexical Analyzer Generator*) é um programa para a geração de analisadores léxicos

Flex

- ▶ Flex (*Fast Lexical Analyzer Generator*) é um programa para a geração de analisadores léxicos
- ▶ Ele foi escrito em linguagem C por Vern Paxson por volta de 1987

Flex

- ▶ Flex (*Fast Lexical Analyzer Generator*) é um programa para a geração de analisadores léxicos
- ▶ Ele foi escrito em linguagem C por Vern Paxson por volta de 1987
- ▶ Ele pode ser usado em conjunto com um gerador de analisadores sintáticos (por exemplo, o Yacc e o GNU Bison)

Flex

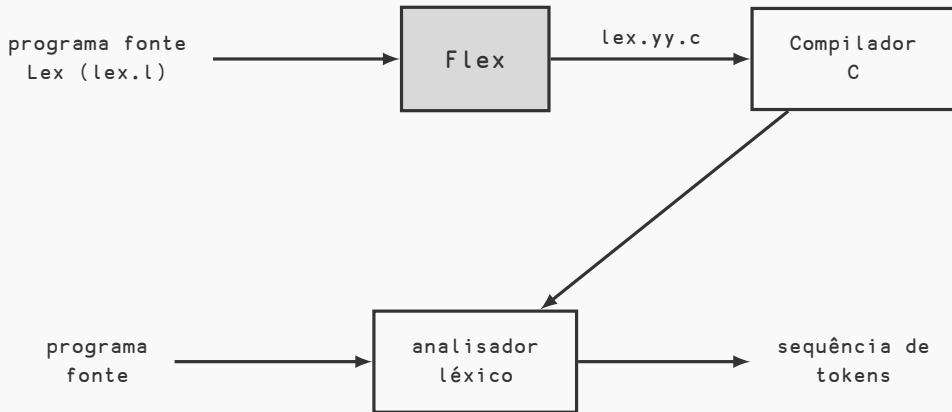
- ▶ Flex (*Fast Lexical Analyzer Generator*) é um programa para a geração de analisadores léxicos
- ▶ Ele foi escrito em linguagem C por Vern Paxson por volta de 1987
- ▶ Ele pode ser usado em conjunto com um gerador de analisadores sintáticos (por exemplo, o Yacc e o GNU Bison)
- ▶ Flex é mais flexível e gera códigos mais rápidos que o Lex, outro programa gerador de analisadores léxicos

Flex

- ▶ Flex (*Fast Lexical Analyzer Generator*) é um programa para a geração de analisadores léxicos
- ▶ Ele foi escrito em linguagem C por Vern Paxson por volta de 1987
- ▶ Ele pode ser usado em conjunto com um gerador de analisadores sintáticos (por exemplo, o Yacc e o GNU Bison)
- ▶ Flex é mais flexível e gera códigos mais rápidos que o Lex, outro programa gerador de analisadores léxicos
- ▶ Ele pode ser instalado, em distribuições Linux baseadas no Debian, por meio do comando

```
$ sudo apt-get install flex
```

Fluxo de uso do Flex para geração de analisadores léxicos



Programas Lex

- ▶ Programas Lex são salvos em arquivos com extensão `.l` (ou `.lex`)

Programas Lex

- ▶ Programas Lex são salvos em arquivos com extensão `.l` (ou `.lex`)
- ▶ Estes programas exportam uma função chamada `yyllex()` que, ao ser chamada, extraí o próximo token do programa fonte

Programas Lex

- ▶ Programas Lex são salvos em arquivos com extensão `.l` (ou `.lex`)
- ▶ Estes programas exportam uma função chamada `yylex()` que, ao ser chamada, extraí o próximo token do programa fonte
- ▶ O código gerado (arquivo `lex.yy.c`) pode ser usado para gerar um executável independente, ou pode ser compilado como código objeto e ser integrado ao analisador sintático

Programas Lex

- ▶ Programas Lex são salvos em arquivos com extensão `.l` (ou `.lex`)
- ▶ Estes programas exportam uma função chamada `yylex()` que, ao ser chamada, extraí o próximo token do programa fonte
- ▶ O código gerado (arquivo `lex.yy.c`) pode ser usado para gerar um executável independente, ou pode ser compilado como código objeto e ser integrado ao analisador sintático
- ▶ Os programas Lex são divididos em três partes: a seção de definições, a seção de regras e a seção de códigos de usuário

Programas Lex

- ▶ Programas Lex são salvos em arquivos com extensão `.l` (ou `.lex`)
- ▶ Este programas exportam uma função chamada `yylex()` que, ao ser chamada, extraí o próximo token do programa fonte
- ▶ O código gerado (arquivo `lex.yy.c`) pode ser usado para gerar um executável independente, ou pode ser compilado como código objeto e ser integrado ao analisador sintático
- ▶ Os programas Lex são divididos em três partes: a seção de definições, a seção de regras e a seção de códigos de usuário
- ▶ A vantagem do uso de programas Lex é que eles permitem a especificação dos tokens por meio de expressões regulares, e a implementação dos diagramas de transição é feita automaticamente pelo Flex

Seção de definições

- ▶ Nesta seção são declaradas variáveis, constantes e definições regulares

Seção de definições

- ▶ Nesta seção são declaradas variáveis, constantes e definições regulares
- ▶ As declarações desta seção deve ser delimitado pelas sequências de caracteres "%{" e "%}"

Seção de definições

- ▶ Nesta seção são declaradas variáveis, constantes e definições regulares
- ▶ As declarações desta seção deve ser delimitado pelas sequências de caracteres "%{" e "%}"
- ▶ O conteúdo desta seção é copiado diretamente para o arquivo `lex.yy.c`

Seção de definições

- ▶ Nesta seção são declaradas variáveis, constantes e definições regulares
- ▶ As declarações desta seção deve ser delimitado pelas sequências de caracteres "%{" e "%}"
- ▶ O conteúdo desta seção é copiado diretamente para o arquivo `lex.yy.c`
- ▶ As definições regulares devem ser declaradas após esta seção, na forma

`nome regex`

Seção de definições

- ▶ Nesta seção são declaradas variáveis, constantes e definições regulares
- ▶ As declarações desta seção deve ser delimitado pelas sequências de caracteres "%{" e "%}"
- ▶ O conteúdo desta seção é copiado diretamente para o arquivo `lex.yy.c`
- ▶ As definições regulares devem ser declaradas após esta seção, na forma

`nome regex`

- ▶ Uma vez definido um nome, ele pode ser usado nas definições regulares subsequentes, desde que sejam delimitados por chaves

Exemplo de seção de declarações

```
1 %{  
2     enum {  
3         LT, LE, EQ, NE, GT, GE,  
4         IF, THEN, ELSE, ID, NUM, RELOP, END_OF_FILE  
5     };  
6  
7     int yylval;  
8     int instalar_id();  
9     int instalar_num();  
10 %}
```

```
12 delim    [ \t\n]  
13 ws       {delim}+  
14 letra    [A-Za-z]  
15 digito    [0-9]  
16 id       {letra}({letra}|{digito})*  
17 num      {digito}+(\.{digito}+)?(E[+-]?{digito}+)?
```

Seção de regras

- ▶ Esta seção contém uma série de regras, uma por linha, na forma
padrão ação

Seção de regras

- ▶ Esta seção contém uma série de regras, uma por linha, na forma
padrão ação
- ▶ O padrão não deve estar indentado e deve estar na mesma linha da ação

Seção de regras

- ▶ Esta seção contém uma série de regras, uma por linha, na forma

padrão ação

- ▶ O padrão não deve estar indentado e deve estar na mesma linha da ação
- ▶ O padrão pode conter algum nome presente nas declarações regulares

Seção de regras

- ▶ Esta seção contém uma série de regras, uma por linha, na forma

padrão ação

- ▶ O padrão não deve estar indentado e deve estar na mesma linha da ação
- ▶ O padrão pode conter algum nome presente nas declarações regulares
- ▶ Neste caso, o nome deve ser delimitado por chaves

Seção de regras

- ▶ Esta seção contém uma série de regras, uma por linha, na forma

padrão ação

- ▶ O padrão não deve estar indentado e deve estar na mesma linha da ação
- ▶ O padrão pode conter algum nome presente nas declarações regulares
- ▶ Neste caso, o nome deve ser delimitado por chaves
- ▶ Esta seção é limitada pela sequência de caracteres %%

Exemplo de seção de declarações

```
19 %%  
20  
21 {ws}      { printf("WS\n"); /* Nenhuma ação e nenhum valor retornado */ }  
22 if        { return IF; }  
23 then      { return THEN; }  
24 else      { return ELSE; }  
25 {id}      { yylval = instalar_id(); return ID; }  
26 {num}     { yylval = instalar_num(); return NUM; }  
27 "<"       { yylval = LT; return RELOP; }  
28 "<="      { yylval = LE; return RELOP; }  
29 "="       { yylval = EQ; return RELOP; }  
30 "<>"      { yylval = NE; return RELOP; }  
31 ">"       { yylval = GT; return RELOP; }  
32 ">="      { yylval = GE; return RELOP; }  
33  
34 %%
```

Seção de código de usuário

- ▶ Esta seção também é copiada diretamente para o arquivo `lex.yy.c`

Seção de código de usuário

- ▶ Esta seção também é copiada diretamente para o arquivo `lex.yy.c`
- ▶ Uma outra alternativa é definir estes códigos em arquivos separados e depois carregar este código na compilação do analisador léxico

Seção de código de usuário

- ▶ Esta seção também é copiada diretamente para o arquivo `lex.yy.c`
- ▶ Uma outra alternativa é definir estes códigos em arquivos separados e depois carregar este código na compilação do analisador léxico

```
36 int instalar_id()
37 {
38     // Insere o lexema e o token na tabela de símbolos e retorna o índice da tabela
39     // onde o símbolo foi inserido. O lexema fica armazenado na variável yytext
40     return -1;
41 }
42
43 int instalar_num()
44 {
45     // Insere o valor do lexema na tabela de números e retorna o índice da tabela
46     // onde o número foi inserido. O lexema fica armazenado na variável yytext
47     return -2;
48 }
```

Exemplo de função `main()` para um analisador léxico independente

```
52 int main()
53 {
54     while (1)
55     {
56         int token = yylex();
57
58         if (token == END_OF_FILE)
59         {
60             printf("Fim da entrada\n");
61             return 0;
62         }
63
64         printf("Token = %d, ytext = %s, yylval = %d\n", token, ytext, yylval);
65     }
66
67     return 0;
68 }
```