TÉCNICO LISBOA

Compiladores[1]

Analisador Lexical Flex

Prof. Alberto Abad

IST - Universidade de Lisboa

2021/2022

[1] Slides adaptados a partir do material do Prof. Pedro T. Monteiro (2017/2018)

Analisador lexical Flex

Flex

Gerador de código que lê um ficheiro contendo uma especificação e gera um analisador lexical (como um módulo C/C++).

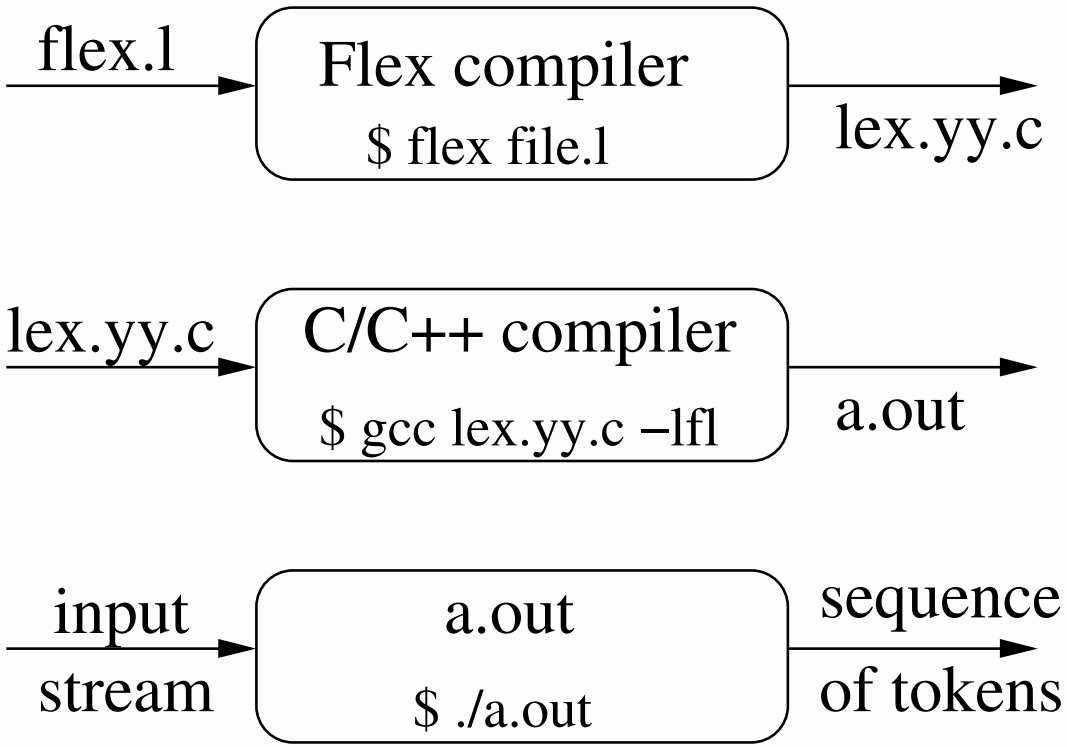
Analisador lexical Flex

Gerador de scanners:

- Lex: desenvolvido na década de 70 para o sistema Unix
  - Proprietário AT&T
- Flex (Fast Lex): desenvolvido nas décadas de 80 e 90
  - Opensource
  - flex (`https://github.com/westes/flex`)
- JFlex, JLex: scanners para Java

Analisador lexical Flex

Etapas



Analisador lexical Flex

Estrutura da especificação

%option {*command_line_options*}
%{
PARTE 1a: preparation code
%}
PARTE 1b: definitions
%%
PARTE 2: rules
%%
PARTE 3: user code

Três secções separadas por uma linha, apenas com os caracteres %%

- Código de preparação é adicionado ao topo do `lex.yy.c`
- Definições e Regras vão definir a função `yylex()` do `lex.yy.c`
- Código do utilizador é adicionado ao fim do `lex.yy.c`

O código de preparação pode incluir:

- includes (e.g. `#include <iostream>`)
- declaração de variáveis globais
- definição de funções auxiliares

As definições podem incluir:

- Definição de símbolos para representar expressões regulares
  e.g. DIGIT [0-9]
  e.g. WHITESPACE [\t]

As definições podem incluir:

- Definição de símbolos para representar expressões regulares
  e.g. DIGIT [0-9]
  e.g. WHITESPACE [\t]
- Start conditions (ou sub-autómatas) para activação condicional de regras

Cada regra é constituída por um par:

Padrão { **Acção** }

- Padrão é uma expressão regular que reconhece uma sequência de caracteres na entrada (pode usar símbolos declarados na secção de definições)
- Acção um conjunto de instruções de código a executar

Exemplo:

{**DIGIT**}+ { std::cout << "Integer: " << yytext << std::endl; }

Selecção de regras:

- Preferência dada a fazer o match da string de **maior** comprimento
- Se a string de maior comprimento representada por mais do que uma RE, escolher a **primeira**

Estratégias:

- Tentar sempre construir o padrão na forma positiva!
- Cuidado a inserir regras e onde
- Cuidado regra *default*

Analisador lexical Flex

Estrutura PARTE 2 - Regras: Expressões regulares

Padrão	Significado
x	match do caracter x
.	match de qualquer caracter, menos fim de linha
(r)	match de r, parentesis servem para ignorar precedência
rs	match r seguido de s
$r \mid s$	match de r ou de s
r^*	zero ou mais r's
r^+	um ou mais r's
$r?$	r opcional
$[xyz]$	match de um dos caracteres x, y ou z
$[abj\text{-}oZ]$	match de um dos caracteres, com intervalo
$[\hat{}A\text{-}Z]$	match qualquer caracter, menos os indicados
$r\{2,5\}$	match de r entre duas 2 cinco vezes
$r\{2,\}$	match the r duas ou mais vezes
$r\{4\}$	match the r exactamente quatro vezes
$\{name\}$	expansão da definição de 'name'
<<EOF>>	match do fim de ficheiro

Variáveis:

- `yytext`: ponteiro para o início do lexema
- `yyleng`: comprimento do lexema
- `yylineno`: linha do input a ser processada
- ...

Analisador lexical Flex

Estrutura PARTE 2 - Funções, variáveis e macros

Variáveis:

- `yytext`: ponteiro para o início do lexema
- `yyleng`: comprimento do lexema
- `yylineno`: linha do input a ser processada
- ...

Funções:

- `yylex()`: função que executa o scanner
- `yymore()`: função que indica ao scanner que o próximo lexema seja concatenado ao actual
- `yyless(n)`: função que devolve ao input todos menos os primeiros *n* caracteres
- ...

Analisador lexical Flex

Estrutura PARTE 2 - Funções, variáveis e macros

Variáveis:

- `yytext`: ponteiro para o início do lexema
- `yyleng`: comprimento do lexema
- `yylineno`: linha do input a ser processada
- ...

Funções:

- `yylex()`: função que executa o scanner
- `yymore()`: função que indica ao scanner que o próximo lexema seja concatenado ao actual
- `yyless(n)`: função que devolve ao input todos menos os primeiros *n* caracteres
- ...

Directivas:

- `ECHO`: copia o conteúdo do `yytext` para o output
- `REJECT`: rejeita a regra actual. O scanner procura a regra seguinte
- ...

Analisador lexical Flex

Estrutura PARTE 3 - Código utilizador

```
...

%%
...

%%
int main() {
    return yylex();
}
```

Analisador lexical Flex

Ficheiro exemplo 1

```
%option noyywrap
%{
#include <iostream>
int num_lines = 0, num_chars = 0;
%}
%%
\n      { ++num_lines; ++num_chars; }
.       { ++num_chars; }
%%
int main() {
    yylex();
    std::cout << "# of lines = " << num_lines << std::endl;
    std::cout << "# of chars = " << num_chars << std::endl;
    return 0;
}
```

Analisador lexical Flex

Ficheiro exemplo 2

```
%option noyywrap
%{
#include <iostream>
int c=0, w=0, l=0;
%}
word [^ \t\n]+
%%
{word} {w++; c+=yyleng;};
\n     {c++; l++;}
.      {c++;}
%%
int main() {
        std::cout << "Vou comecar:" << std::endl;
    yylex();
    std::cout << "# of lines = " << l << std::endl;
    std::cout << "# of words = " << w << std::endl;
    std::cout << "# of chars = " << c << std::endl;
    return 0;
}
```

Analisador lexical Flex

Opções - yywrap

Quando o EOF é lido na entrada, o flex verifica a função `yywrap()`

- Se devolver 0, o próximo ficheiro vai ser lido
- Caso contrário, o scanner termina

Caso o utilizador não defina a função `yywrap()`

- Acrescentar `%option noyywrap`, para usar a função por omissão que devolve sempre 1

Analisador lexical Flex

Opções - yylineno

Para que o flex gere um scanner que mantenha o número da linha da entrada numa variável global `yylineno`

- Acrescentar `%option yylineno`

Analisador lexical Flex

Opções - 8bit

Para que o flex gere um scanner de 8 bits

- Acrescentar `%option 8bit`

Para uma entrada ASCII, um scanner de 7 bits é suficiente

Analisador lexical Flex

Opções - debug

- Acrescentar `%option debug`
- Em C++ também é preciso acrescentar a ação `set_debug(1)` ao inicio da seção de regras:

```
%option debug
...

%%
          { set_debug(1); }

... other rules ...

%%
```

Analisador lexical Flex

Opções - stack

Para que o flex gere um scanner com suporte de pilha para as start conditions

- Acrescentar `%option stack`

Start conditions

Mecanismo para ativar regras condicionais (ou sub-autómatas):

- Uma start condition é activada através da acção BEGIN(new_state)
- Regras com uma dada condição estão activas e todas as outras estão inactivas, até que a próxima acção BEGIN() seja executada
- INITIAL (ou vazia) é a start condition por omissão

Start conditions - Tipos

- inclusivas: precedidas por `%s`
  considera todas as regras precedidas pela start condition e não precedidas por nenhuma
- exclusivas: precedidas por `%x`
  considera apenas as regras precedidas pela start condition

Start conditions - Definição

- `<COND1>regexp` regra para a COND1
- `<COND1,COND2>regexp` regra para COND1 e COND2
- `<*>regexp` regra para todas as start condition (incluida inicial)

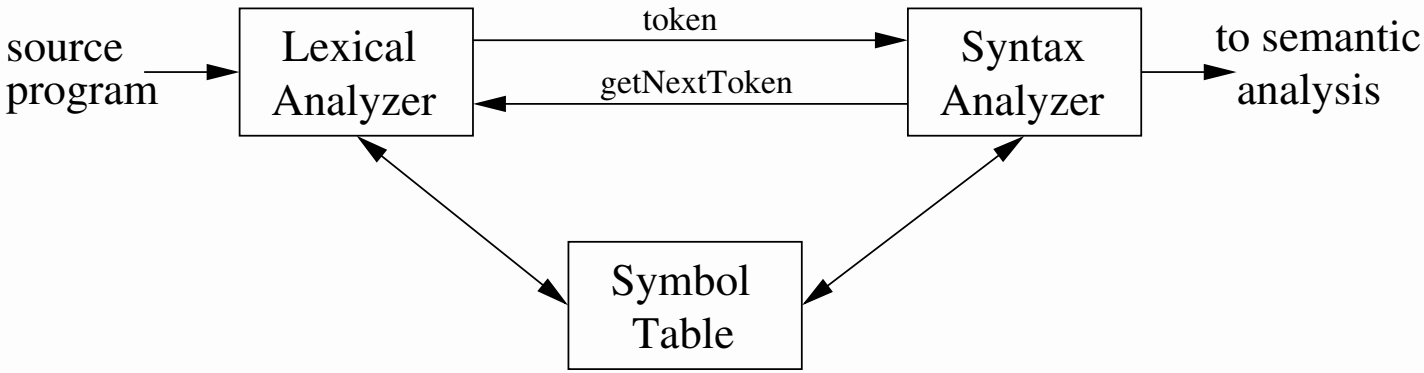
Start conditions - Pilha

Flex fornece uma pilha para as stack conditions

- `yy_push_state(new_state)` - guarda a start condition atual no topo da pilha e muda o seu estado para new_state (como BEGIN(new_state)).
- `yy_pop_state()` - retira a start condition do topo da pilha e muda para ela via BEGIN

Analisador lexical Flex

Ligação com o analisador sintáctico



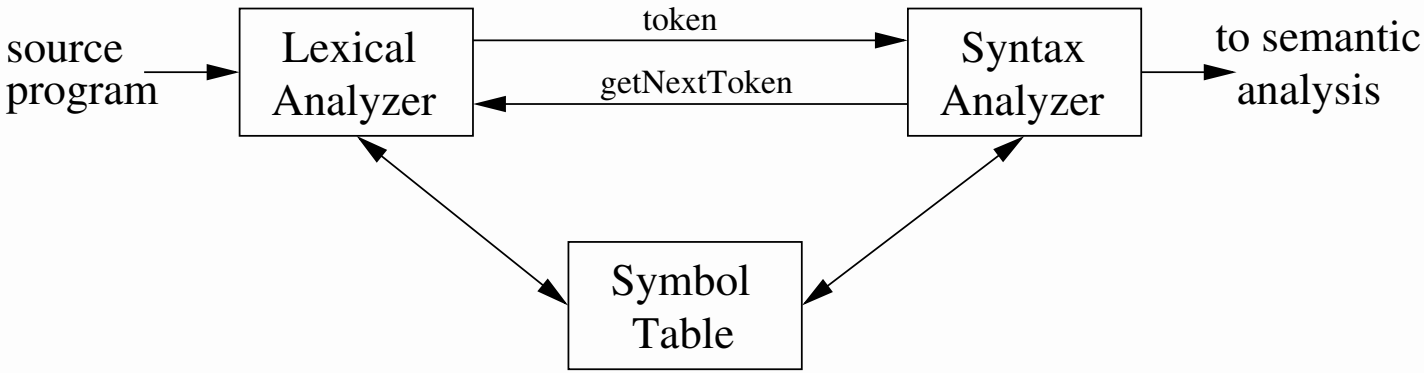
Tokens:

- Parser define tokens da linguagem
- Scanner inclui ficheiro com definição dos tokens
- No scanner, cada padrão pode retornar um token/lexema
  - Última instrução da Acção é `return TOKENNAME;`

```
while ((tok = yylex()) != 0)
 /* do something...  */ ;
```

Analisador lexical Flex

Ligação com o analisador sintáctico



Por vezes é necessário conhecer o valor do lexema reconhecido

- O valor do atributo associado a um token é comunicado ao parser através da variável `yylval` definida pelo YACC
- Devido à existência de vários tipos de dados, `yylval` é uma *union*
  - Cada token corresponde apenas a um tipo

Exemplo:

```
[a-z]+        { yylval.s = new std::string(yytext); return tID; }
```

Exemplos

WIKI

https://web.tecnico.ulisboa.pt/~david.matos/w/pt/index.php/The_Flex_Lexical_Analyzer

Questões?

Dúvidas?