Compiladores

Ficha prática 3 – Lex (continuação)

O *lex* permite o acesso a um conjunto de variáveis e funções pré-definidas, de entre as quais destacamos:

Nome	**Descrição**
`int yylex(void)`	Chamada do analisador lexical(entry point)
`char *yytext`	Ponteiro para a string detectada (o token)
`yyleng`	Comprimento da string detectada (do token)
`yylval`	Valor associado ao token
`int yywrap(void)`	Wrapup, devolve 1 se acabou input, 0 caso contrário
`int yylineno`	Número de linha do ficheiro de entrada
`FILE *yyin`	Ponteiro para ficheiro de entrada
`FILE *yyout`	Ponteiro para ficheiro de saida
`BEGIN`	Macro para condição inicial no switch
`ECHO`	Macro para repetir na saída a string detectada

Por agora, vamos utilizar o ponteiro para a string (*yytext*). Com esta variável, temos acesso directo ao token (ou padrão), e portanto são muito maiores as potencialidades do *lex*. Por exemplo, se, em vez de escrevermos simplesmente "Inteiro", quiséssemos substituir o número pelo seu equivalente em hexadecimal, a regra seria:

```
%%
[0-9]+                  {printf("%X", atoi(yytext));}
```

Lembramos que "%X" imprime um inteiro em hexadecimal (na função printf, de "C") e que a função `atoi`(char*) devolve o valor inteiro contido numa string.

Outra possibilidade no *lex* é a declaração de variáveis na parte de definições (desde que entre %{ e %}). No código seguinte, podemos ver um exemplo da sua aplicação:

```
%{
int numints=0;          /*Contador de inteiros*/
%}
%%
[0-9]+                  {printf("%X", atoi(yytext));/* Passa a hex*/
                         numints++;}                /* Soma um */
[0-9]+"."[0-9]+         ;     /*Real, não faz nada */
.                       ;     /*Tudo o resto excepto new line */
\n                      ;     /*new line*/


%%
int main()
{
```

```
yylex();
printf("Numero de inteiros=%d", numints); /*Mostra resultado*/
return 0;
}

int yywrap()
{
return 1;
}
```

1. Faça um analisador lexical que transforme, no texto de input, todas as minúsculas em maiúsculas (e mantenha tudo o resto igual).

2. Faça um analisador lexical que extrai de um documento de texto todos os endereços de email contidos. Assuma que os caracteres não permitidos num endereço de email são "\t", "\n", "{", "}", *espaço, ponto e vírgula, vírgula* e *aspas*. Lembre-se também que não pode haver dois "@" nem dois *pontos* seguidos. Este analisador deve apenas produzir a lista de endereços, separados de vírgula e espaço. Ou seja, prontos para serem copiados directamente para um cliente de email. Por exemplo, o texto:

```
"O endereço do João Fonseca é jfonseca@gmail.com e o do Carlos
Meireles é meiras@hotmail.com. Para os outros, estão na lista abaixo:
- mariliaf@student.deeei.uq.tp; mfon{s}@kkks.us
- gilaroi@yaaahooo.com; gil eanes@grlah.jh"
```

daria o seguinte resultado

```
jfonseca@gmail.com, meiras@hotmail.com, mariliaf@student.deeei.uq.tp,
gilaroi@yaaahooo.com, eanes@grlah.jh
```

3. Dado um ficheiro .srt (de legendas para Divx), fazer um analisador lexical para adiantar/atrasar essas legendas de acordo com um valor de *x* segundos.

Exemplo do formato de um ficheiro .srt:
1
00:00:45,000 --> 00:00:50,400
Hello

2
00:00:50,300 --> 00:00:58,200
Goodbye

Neste exemplo serão geradas duas legendas. A primeira, será apresentada no filme aos 45 segundos durante cerca de 5 segundos. A segunda, será apresentada no filme aos 50 segundos durante cerca de 3 segundos.

Bibliografia recomendada:
. Anexo A de *Processadores de Linguagens*. Rui Gustavo Crespo. IST Press. 1998
. *A Compact Guide to Lex & Yacc*. T. Niemann. http://epaperpress.com/lexandyacc/ epaperpress.
. Manual do lex/flex em Unix (comando "man lex" na shell)