

Join GitHub today

Dismiss

GitHub is home to over 28 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: dev ▼ [compiler](#) / [lab2](#) / [tiger.lex](#)[Find file](#)[Copy path](#)

Cannot retrieve latest commit at this time.

[0 contributors](#)

230 lines (180 sloc) 4.93 KB

```
1  %{
2  #include <string.h>
3  #include "util.h"
4  #include "tokens.h"
5  #include "errmsg.h"
6
7  int charPos=1;
8
9  int yywrap(void)
10 {
11     charPos=1;
12     return 1;
13 }
14
15 void adjust(void)
16 {
17     EM_tokPos=charPos;
18     charPos+=yyleng;
19 }
20
21 /*
22  * Please don't modify the lines above.
23  * You can add C declarations of your own below.
24  */
25
26 #define MAX_STR_LEN 1024
27
28 int commentLevel=0; /* for nested comment */
29
30 char string_buf[MAX_STR_LEN + 1];
31 char *string_buf_ptr;
32
33 void adjuststr(void)
34 {
35     charPos+=yyleng;
36 }
37
38
39 /* @function: getstr
40  * @input: a string literal
41  * @output: the string value for the input which has all the escape sequences
42  * translated into their meaning.
43  */
44 char *getstr(const char *str)
45 {
46     return NULL;
47 }
48
49 int chtodec (char c)
50 {
51     if( c>='@' && c<='Z' ){
52         c -= 64; //convert\^A(\^065)to\^001
```

```

53     }else{
54         EM_error(charPos, "Unknown Controll Character.");
55     }
56     return c;
57 }
58
59
60 unsigned long charCount = 0, wordCount = 0, lineCount = 0;
61
62 #undef yywrap /* sometimes a macro by default */
63
64 %}
65 /* You can add lex definitions here. */
66
67 %x str comment
68
69 INTregExp [0-9]+
70 IDregExp [a-zA-Z][a-zA-Z0-9_]*
71
72
73 %%
74 /*
75  * Below are some examples, which you can wipe out
76  * and write regular expressions and actions of your own.
77
78  */
79
80
81 /* string */
82 <str>{
83
84     \" {
85         adjuststr();
86         *string_buf_ptr='\0';
87         if (string_buf[0] != '\0')
88             yylval.sval=String(string_buf);
89         else
90             yylval.sval=String("(null)"); /* Compatible with test case */
91         BEGIN(INITIAL);
92         return STRING;
93     } /* comment starting */
94
95     \\([0-9]{3}) {
96         adjuststr();
97         int result = atoi(yytext + 1);
98         char c = (char)result;
99
100         if (result > 0xff) {
101             EM_error(EM_tokPos, "illegal character");
102             continue;
103         }
104         *string_buf_ptr++ = c;
105     }
106
107     \\n    {adjuststr(); *string_buf_ptr++ = '\n';}
108     \\t    {adjuststr(); *string_buf_ptr++ = '\t';}
109     \\\\"   {adjuststr(); *string_buf_ptr++ = '\\';}
110     \\\\    {adjuststr(); *string_buf_ptr++ = '\\\\';}
111     \\^[\\0-\\037] {
112         adjuststr();
113         *string_buf_ptr++ = yytext[2];
114     } /* \\^[ means "\\^" OCT:\0-\37 */
115
116
117
118     (\\\\^)[\x41-\x5A\x61-\x7A] {
119         adjuststr();
120         *string_buf_ptr++ = chtodec(yytext[2]);
121     } /* \\^[ means "\\^", OCT:\101-\132\141-\172, Hex:\x41-\x5A\x61-\x7A Dec:65-90,97-122 means A~Za~z Ex. char ch=67, char *st
122
123
124     \\[ \\t\\n\\r]+\\ {
125         adjuststr();

```

```

126     char *yytextptr = yytext;
127     while (*yytextptr != '\0')
128     {
129         if (*yytextptr == '\n')
130             EM_newline();
131         ++yytextptr;
132     }
133 }
134
135 \\. {adjuststr(); EM_error(charPos, "illegal escape char");}
136
137 \n {
138     adjuststr();
139     EM_newline();
140     EM_error(charPos, "string terminated with newline");
141     continue;
142 }
143
144 [^\\n\\"]+ {
145     adjuststr();
146     char *yptr = yytext;
147
148     while (*yptr)
149         *string_buf_ptr++ = *yptr++;
150 }
151 }
152
153 /* comment, note that the special start-condition specifier `<*>' matches every start condition.
154    Can match where ever < > state is in .
155 */
156
157 <*>{
158     "/*" {adjust(); ++commentLevel; BEGIN(comment);}
159 }
160
161 <comment>{
162     \n {adjust(); EM_newline();}
163     "*/" {adjust(); --commentLevel; if (commentLevel <= 0) BEGIN(INITIAL);}
164     . {adjust();}
165 }
166
167 <INITIAL>{
168
169     (" "|"\\t") {adjust();}
170     \n {adjust(); EM_newline();}
171
172     "," {adjust(); return COMMA;}
173     ":" {adjust(); return COLON;}
174     ";" {adjust(); return SEMICOLON;}
175
176     "(" {adjust(); return LPAREN;}
177     ")" {adjust(); return RPAREN;}
178     "[" {adjust(); return LBRACK;}
179     "]" {adjust(); return RBRACK;}
180     "{" {adjust(); return LBRACE;}
181     "}" {adjust(); return RBRACE;}
182
183     "." {adjust(); return DOT;}
184     "+" {adjust(); return PLUS;}
185     "-" {adjust(); return MINUS;}
186     "*" {adjust(); return TIMES;}
187     "/" {adjust(); return DIVIDE;}
188
189     "=" {adjust(); return EQ;}
190     "<>" {adjust(); return NEQ;}
191     "<" {adjust(); return LT;}
192     "<=" {adjust(); return LE;}
193     ">" {adjust(); return GT;}
194     ">=" {adjust(); return GE;}
195     "&" {adjust(); return AND;}
196     "|" {adjust(); return OR;}
197     "!=" {adjust(); return ASSIGN;}
198

```

```
199 array {adjust(); return ARRAY;}
200 if {adjust(); return IF;}
201 then {adjust(); return THEN;}
202 else {adjust(); return ELSE;}
203 while {adjust(); return WHILE;}
204 for {adjust(); return FOR;}
205 to {adjust(); return TO;}
206 do {adjust(); return DO;}
207 let {adjust(); return LET;}
208 in {adjust(); return IN;}
209 end {adjust(); return END;}
210 of {adjust(); return OF;}
211 break {adjust(); return BREAK;}
212 nil {adjust(); return NIL;}
213 function {adjust(); return FUNCTION;}
214 var {adjust(); return VAR;}
215 type {adjust(); return TYPE;}
216
217
218 {INTregExp} {adjust(); yylval.ival=atoi(yytext); return INT;}
219 {IDregExp} {adjust(); yylval.sval=String(yytext); return ID;}
220 \" {
221     adjust();
222     string_buf_ptr = string_buf;
223     BEGIN(str);
224 }
225 }
226
227 . {adjust(); EM_error(EM_tokPos,\"illegal token");}
228
229 %%
```