```
%{
////////////////////////////////////////////////////////////////////////////////
//Class:          CS 445
//Semester:       Fall 2011
//Assignment:     Homework 4
//Author:         Colby Blair
//File name:      AS3.l
////////////////////////////////////////////////////////////////////////////////


#include "parser.tab.h"
#include "main.h"
#include "tree.h"

//#define DEBUG_PARSER

//macro for the tree_create_leaf argument
#ifdef DEBUG_PARSER
#define _TLA(CATEGORY)  DEBUGMSG("DEBUG: %d - %s \t\t\tline: %d\n", CATEGORY, yytext
, lineno); yylval.t = tree_create_node_from_token((int)CATEGORY, yytext, lineno, YY_
FNAME); colno += strlen(yytext); return( (int)CATEGORY );
#else
#define _TLA(CATEGORY)  yylval.t = tree_create_node_from_token((int)CATEGORY, yytext
, lineno, YY_FNAME); colno += strlen(yytext); return( (int)CATEGORY );
#endif

int lineno;
int colno;
extern char *YY_FNAME;


int newline_count(char s[])
{
        int sum = 0;
        int i;
        for(i = 0; i < strlen(s); i++)
        {
                if(s[i] == '\n')
                {
                        sum++;
                }
        }

        return(sum);
}

%}

id              [a-zA-Z_]+[.a-zA-Z0-9_]*
decimal_literal [\-]{0,1}[0-9]+|[\-]{0,1}[0-9]+"."[0-9]+

string_lit      "\""[^\"\n]*"\""|"'"[^\"\n]*"'"



%%
"//".*$                 { }
"/*"[^\*\/]*"*/"        { lineno += newline_count(yytext); }
"\n"                    { ++lineno; colno = 1; }
" "                     { /*white space, no op for now*/ }
"\t"                    { /*white space, no op for now*/ }

"as"                    { _TLA(AS ) }
"break"                 { _TLA(BREAK ) }
"case"                  { _TLA(CASE ) }
"catch"                 { _TLA(CATCH ) }
"class"                 { _TLA(CLASS ) }
"const"                 { _TLA(CONST ) }
"continue"              { _TLA(CONTINUE ) }
"default"               { _TLA(DEFAULT ) }
"delete"                { _TLA(DELETE ) }
"do"                    { _TLA(DO ) }
"else"                  { _TLA(ELSE ) }
```

```
"extends"               { _TLA(EXTENDS ) }
"false"                 { _TLA(FALSE ) }
"finally"               { _TLA(FINALLY ) }
"for"                   { _TLA(FOR ) }
"function"              { _TLA(FUNCTION ) }
"if"                    { _TLA(IF ) }
"implements"            { _TLA(IMPLEMENTS ) }
"import"                { _TLA(IMPORT ) }
"in"                    { _TLA(IN ) }
"instanceof"            { _TLA(INSTANCEOF ) }
"interface"             { _TLA(INTERFACE ) }
"internal"              { _TLA(INTERNAL ) }
"is"                    { _TLA(IS ) }
"new"                   { _TLA(NEW ) }
"null"|"Null"           { _TLA(NULL_VAL) }
"package"               { _TLA(PACKAGE ) }
"private"               { _TLA(PRIVATE ) }
"protected"             { _TLA(PROTECTED ) }
"public"                { _TLA(PUBLIC ) }
"return"                { _TLA(RETURN ) }
"super"                 { _TLA(SUPER ) }
"switch"                { _TLA(SWITCH ) }
"throw"                 { _TLA(THROW ) }
"true"                  { _TLA(TRUE ) }
"try"                   { _TLA(TRY ) }
"typeof"                { _TLA(TYPEOF ) }
"use"                   { _TLA(USE ) }
"var"                   { _TLA(VAR ) }
"void"                  { _TLA(VOID ) }
"while"                 { _TLA(WHILE ) }
"with"                  { _TLA(WITH ) }

"each"                  { _TLA(EACH ) }
"get"                   { _TLA(GET ) }
"set"                   { _TLA(SET ) }
"namespace"             { _TLA(NAMESPACE ) }
"include"               { _TLA(INCLUDE ) }
"dynamic"               { _TLA(DYNAMIC ) }
"final"                 { _TLA(FINAL ) }
"override"              { _TLA(OVERRIDE ) }
"static"                { _TLA(STATIC ) }

";"                     { _TLA(SEMI) }
"="                     { _TLA(ASSIGN) }
","                     { _TLA(COMMA) }
"/"                     { _TLA(DIV) }
"["                     { _TLA(LBRACK) }
"]"                     { _TLA(RBRACK) }
"{"                     { _TLA(LCURLY) }
"}"                     { _TLA(RCURLY) }
"("                     { _TLA(LPAREN) }
")"                     { _TLA(RPAREN) }
"?"                     { _TLA(QUESTION) }
":"                     { _TLA(COLON) }
"@"                     { _TLA(E4X_ATTRI) }
"&"                     { _TLA(BAND) }
"+"                     { _TLA(PLUS) }
"-"                     { _TLA(MINUS) }
"<"                     { _TLA(LT) }
">"                     { _TLA(GT) }
"<="                    { _TLA(LE) }
">="                    { _TLA(GE) }
"=="                    { _TLA(EQUAL) }
"!="                    { _TLA(NOT_EQUAL) }
"==="                   { _TLA(STRICT_EQUAL) }
"!=="                   { _TLA(STRICT_NOT_EQUAL) }
"*"                     { _TLA(STAR) }
"%"                     { _TLA(MOD) }
"++"                    { _TLA(INC) }
"--"                    { _TLA(DEC) }
"<<"                    { _TLA(SL) }
">>"                    { _TLA(SR) }
">>>"                   { _TLA(SL_ASSIGN) }
```

```
"|"                         { _TLA(BOR) }
"^"                         { _TLA(BXOR) }
"!"                         { _TLA(LNOT) }
"~"                         { _TLA(BNOT) }
"&&"                        { _TLA(LAND) }
"||"                        { _TLA(LOR) }
"+="                        { _TLA(PLUS_ASSIGN) }
"-="                        { _TLA(MINUS_ASSIGN) }
"*="                        { _TLA(STAR_ASSIGN) }
"%="                        { _TLA(MOD_ASSIGN) }
"<<="                       { _TLA(SL_ASSIGN) }
">>="                       { _TLA(SR_ASSIGN) }
">>>="                      { _TLA(BSR_ASSIGN) }
"&="                        { _TLA(BAND_ASSIGN) }
"|="                        { _TLA(BOR_ASSIGN) }
"^="                        { _TLA(BXOR_ASSIGN) }

{string_lit}                { _TLA(STRING_LITERAL) }
{id}                        { _TLA(IDENT) }
{decimal_literal}           { _TLA(DECIMAL_LITERAL) }

<<EOF>>                     { return(EOFX); }

%%
```