ANSI C grammar, Lex specification

(This Lex file is accompanied by a matching Yacc file.)

In 1985, Jeff Lee published his Yacc grammar based on a draft version of the ANSI C standard, along with a supporting Lex specification. Tom Stockfisch reposted those files to net.sources in 1987; as mentioned in the answer to question 17.25 of the comp.lang.c FAQ, they used to be available from ftp.uu.net as usenet/net.sources/ansi.c.grammar.Z.

The version you see here has been updated based on the 2011 ISO C standard. (The previous version's Lex and Yacc files for ANSI C9X still exist as archived copies.)

It is assumed that translation phases 1..5 have already been completed, including preprocessing and `_Pragma` processing. The Lex rule for string literals will perform concatenation (translation phase 6). Transliteration of universal character names (`\u`*HHHH* or `\U`*HHHHHHHH*) must have been done by either the preprocessor or a replacement for the `input()` macro used by Lex (or the YY_INPUT function used by Flex) to read characters. Although comments should have been changed to space characters during translation phase 3, there are Lex rules for them anyway.

I want to keep this version as close to the current C Standard grammar as possible; please let me know if you discover discrepancies.
(There is an FAQ for this grammar that you might want to read first.)

jutta@pobox.com, 2012

Last edit: 2012-12-19 DAGwyn@aol.com

Note: The following %-parameters are the minimum sizes needed for real Lex.

`%e` number of parsed tree nodes

`%p` number of positions

`%n` number of states

`%k` number of packed character classes

%a number of transitions

%o size of output array

```
%e  1019
%p  2807
%n  371
%k  284
%a  1213
%o  1117

O   [0-7]
D   [0-9]
NZ  [1-9]
L   [a-zA-Z_]
A   [a-zA-Z_0-9]
H   [a-fA-F0-9]
HP  (0[xX])
E   ([Ee][+-]?{D}+)
P   ([Pp][+-]?{D}+)
FS  (f|F|l|L)
IS  (((u|U)(l|L|ll|LL)?)|((l|L|ll|LL)(u|U)?))
CP  (u|U|L)
SP  (u8|u|U|L)
ES  (\\(['"\?\\abfnrtv]|[0-7]{1,3}|x[a-fA-F0-9]+))
WS  [ \t\v\n\f]

%{
#include <stdio.h>
#include "y.tab.h"

extern void yyerror(const char *);  /* prints grammar violation message */

extern int sym_type(const char *);  /* returns type from symbol table */

#define sym_type(identifier) IDENTIFIER /* with no symbol table, fake it */

static void comment(void);
static int check_type(void);
%}

%%
"/*"                                    { comment(); }
"//".*                                  { /* consume //-comment */ }

"auto"                                  { return(AUTO); }
"break"                                 { return(BREAK); }
"case"                                  { return(CASE); }
"char"                                  { return(CHAR); }
"const"                                 { return(CONST); }
"continue"                              { return(CONTINUE); }
"default"                               { return(DEFAULT); }
"do"                                    { return(DO); }
"double"                                { return(DOUBLE); }
"else"                                  { return(ELSE); }
"enum"                                  { return(ENUM); }
"extern"                                { return(EXTERN); }
"float"                                 { return(FLOAT); }
```

```
"for"                                   { return(FOR); }
"goto"                                  { return(GOTO); }
"if"                                    { return(IF); }
"inline"                                { return(INLINE); }
"int"                                   { return(INT); }
"long"                                  { return(LONG); }
"register"                              { return(REGISTER); }
"restrict"                              { return(RESTRICT); }
"return"                                { return(RETURN); }
"short"                                 { return(SHORT); }
"signed"                                { return(SIGNED); }
"sizeof"                                { return(SIZEOF); }
"static"                                { return(STATIC); }
"struct"                                { return(STRUCT); }
"switch"                                { return(SWITCH); }
"typedef"                               { return(TYPEDEF); }
"union"                                 { return(UNION); }
"unsigned"                              { return(UNSIGNED); }
"void"                                  { return(VOID); }
"volatile"                              { return(VOLATILE); }
"while"                                 { return(WHILE); }
"_Alignas"                              { return ALIGNAS; }
"_Alignof"                              { return ALIGNOF; }
"_Atomic"                               { return ATOMIC; }
"_Bool"                                 { return BOOL; }
"_Complex"                              { return COMPLEX; }
"_Generic"                              { return GENERIC; }
"_Imaginary"                            { return IMAGINARY; }
"_Noreturn"                             { return NORETURN; }
"_Static_assert"                        { return STATIC_ASSERT; }
"_Thread_local"                         { return THREAD_LOCAL; }
"__func__"                              { return FUNC_NAME; }

{L}{A}*                                 { return check_type(); }

{HP}{H}+{IS}?                           { return I_CONSTANT; }
{NZ}{D}*{IS}?                           { return I_CONSTANT; }
"0"{O}*{IS}?                            { return I_CONSTANT; }
{CP}?"'"([^'\\\n]|{ES})+"'"             { return I_CONSTANT; }

{D}+{E}{FS}?                            { return F_CONSTANT; }
{D}*"."{D}+{E}?{FS}?                    { return F_CONSTANT; }
{D}+"."{E}?{FS}?                        { return F_CONSTANT; }
{HP}{H}+{P}{FS}?                        { return F_CONSTANT; }
{HP}{H}*"."{H}+{P}{FS}?                 { return F_CONSTANT; }
{HP}{H}+"."{P}{FS}?                     { return F_CONSTANT; }

({SP}?\"([^"\\\n]|{ES})*\"{WS}*)+       { return STRING_LITERAL; }

"..."                                   { return ELLIPSIS; }
">>="                                   { return RIGHT_ASSIGN; }
"<<="                                   { return LEFT_ASSIGN; }
"+="                                    { return ADD_ASSIGN; }
"-="                                    { return SUB_ASSIGN; }
"*="                                    { return MUL_ASSIGN; }
"/="                                    { return DIV_ASSIGN; }
"%="                                    { return MOD_ASSIGN; }
"&="                                    { return AND_ASSIGN; }
"^="                                    { return XOR_ASSIGN; }
```

```
"|="                                    { return OR_ASSIGN; }
">>"                                    { return RIGHT_OP; }
"<<"                                    { return LEFT_OP; }
"++"                                    { return INC_OP; }
"--"                                    { return DEC_OP; }
"->"                                    { return PTR_OP; }
"&&"                                    { return AND_OP; }
"||"                                    { return OR_OP; }
"<="                                    { return LE_OP; }
">="                                    { return GE_OP; }
"=="                                    { return EQ_OP; }
"!="                                    { return NE_OP; }
";"                                     { return ';'; }
("{"|"<%")                              { return '{'; }
("}"|"%>")                              { return '}'; }
","                                     { return ','; }
":"                                     { return ':'; }
"="                                     { return '='; }
"("                                     { return '('; }
")"                                     { return ')'; }
("["|"<:")                              { return '['; }
("]"|":>")                              { return ']'; }
"."                                     { return '.'; }
"&"                                     { return '&'; }
"!"                                     { return '!'; }
"~"                                     { return '~'; }
"-"                                     { return '-'; }
"+"                                     { return '+'; }
"*"                                     { return '*'; }
"/"                                     { return '/'; }
"%"                                     { return '%'; }
"<"                                     { return '<'; }
">"                                     { return '>'; }
"^"                                     { return '^'; }
"|"                                     { return '|'; }
"?"                                     { return '?'; }

{WS}+                                   { /* whitespace separates tokens */ }
.                                       { /* discard bad characters */ }

%%

int yywrap(void)        /* called at end of input */
{
    return 1;           /* terminate now */
}

static void comment(void)
{
    int c;

    while ((c = input()) != 0)
        if (c == '*')
        {
            while ((c = input()) == '*')
                ;

            if (c == '/')
                return;
```

```
            if (c == 0)
                break;
        }
    yyerror("unterminated comment");
}

static int check_type(void)
{
    switch (sym_type(yytext))
    {
    case TYPEDEF_NAME:                /* previously defined */
        return TYPEDEF_NAME;
    case ENUMERATION_CONSTANT:        /* previously defined */
        return ENUMERATION_CONSTANT;
    default:                          /* includes undefined */
        return IDENTIFIER;
    }
}
```