

```
/*WARNING: This file is automatically generated!*/  
/* YACC parser for C syntax and for Objective C. -*-c-*/  
Copyright (C) 1987, 1988, 1989, 1992, 1993, 1994, 1995, 1996,  
1997, 1998, 1999, 2000, 2001 Free Software Foundation, Inc.
```

This file is part of GCC.

GCC is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

GCC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

*You should have received a copy of the GNU General Public License along with GCC; see the file COPYING. If not, write to the Free Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. */*

```
/* This file defines the grammar of C and that of Objective C.  
ifobjc ... end ifobjc conditionals contain code for Objective C only.  
ifc ... end ifc conditionals contain code for C only.  
Sed commands in Makefile.in are used to convert this file into  
c-parse.y and into objc-parse.y. */
```

```
/* To whomever it may concern: I have heard that such a thing was once  
written by AT&T, but I have never seen it. */
```

```
%expect 10 /* shift/reduce conflicts, and no reduce/reduce conflicts. */
```

```
%start program
```

```
/* All identifiers that are not reserved words  
and are not declared typedefs in the current block */  
%token IDENTIFIER
```

```
/* All identifiers that are declared typedefs in the current block.  
In some contexts, they are treated just like IDENTIFIER,  
but they can also serve as typespecs in declarations. */  
%token TYPENAME
```

```
/* Reserved words that specify storage class.  
yyval contains an IDENTIFIER_NODE which indicates which one. */  
%token SCSPEC
```

```
/* Reserved words that specify type.  
yyval contains an IDENTIFIER_NODE which indicates which one. */  
%token TYPESPEC
```

```
/* Reserved words that qualify type: "const", "volatile", or "restrict".  
yyval contains an IDENTIFIER_NODE which indicates which one. */  
%token TYPE_QUAL
```

```
/* Character or numeric constants.  
yyval is the node for the constant. */
```

```

%token CONSTANT

/* String constants in raw form.
   yyval is a STRING_CST node. */
%token STRING

/* "...", used for functions with variable arglists. */
%token ELLIPSIS

/* the reserved words */
/* SCO include files test "ASM", so use something else. */
%token SIZEOF ENUM STRUCT UNION IF ELSE WHILE DO FOR SWITCH CASE DEFAULT
%token BREAK CONTINUE RETURN GOTO ASM_KEYWORD TYPEOF ALIGNOF
%token ATTRIBUTE EXTENSION LABEL
%token REALPART IMAGPART VA_ARG CHOOSE_EXPR TYPES_COMPATIBLE_P
%token PTR_VALUE PTR_BASE PTR_EXTENT

/* function name can be a string const or a var decl. */
%token STRING_FUNC_NAME VAR_FUNC_NAME

/* Add precedence rules to solve dangling else s/r conflict */
%nonassoc IF
%nonassoc ELSE

/* Define the operator tokens and their precedences.
   The value is an integer because, if used, it is the tree code
   to use in the expression made from the operator. */

%right <code> ASSIGN '='
%right <code> '?' ':'
%left <code> OROR
%left <code> ANDAND
%left <code> '|'
%left <code> '^'
%left <code> '&'
%left <code> EQCOMPARE
%left <code> ARITHCOMPARE
%left <code> LSHIFT RSHIFT
%left <code> '+' '-'
%left <code> '*' '/' '%'
%right <code> UNARY PLUSPLUS MINUSMINUS
%left HYPERUNARY
%left <code> POINTSAT '.' '(' '['

/* The Objective-C keywords. These are included in C and in
   Objective C, so that the token codes are the same in both. */
%token INTERFACE IMPLEMENTATION END_SELECTOR DEFS ENCODE
%token CLASSNAME PUBLIC PRIVATE PROTECTED PROTOCOL OBJECTNAME CLASS ALIAS

%%
program: /* empty */
        | extdefs
        ;

/* the reason for the strange actions in this rule
   is so that notype_initdecls when reached via datadef
   can find a valid list of type and sc specs in $0. */

```

```

extdefs: extdef
      | extdefs extdef
      ;

extdef:
      fndef
      | datadef
      | ASM_KEYWORD '(' expr ')' ';'
      | extension extdef
      ;

datadef:
      setspecs notype_initdecls ';'
      | declspecs_nots setspecs notype_initdecls ';'
      | declspecs_ts setspecs initdecls ';'
      | declspecs ';'
      | error ';'
      | error '}'
      | ';'
      ;

fndef:
      declspecs_ts setspecs declarator old_style_parm_decls save_filename save_lineno compstmt_or_error
      | declspecs_ts setspecs declarator error
      | declspecs_nots setspecs notype_declarator old_style_parm_decls save_filename save_lineno compstmt_or_error
      | declspecs_nots setspecs notype_declarator error
      | setspecs notype_declarator old_style_parm_decls save_filename save_lineno compstmt_or_error
      | setspecs notype_declarator error
      ;

identifier:
      IDENTIFIER
      | TYPENAME
      ;

unop:
      '&'
      | '-'
      | '+'
      | PLUSPLUS
      | MINUSMINUS
      | '~'
      | '!'
      ;

expr:
      nonnull_explist
      ;

explist:
      /* empty */
      | nonnull_explist
      ;

nonnull_explist:
      expr_no_commas
      | nonnull_explist ',' expr_no_commas
      ;

unary_expr:

```

```

primary
| '*' cast_expr %prec UNARY
/* __extension__ turns off -pedantic for following primary. */
| extension cast_expr %prec UNARY
| unop cast_expr %prec UNARY
/* Refer to the address of a label as a pointer. */
| ANDAND identifier
| sizeof unary_expr %prec UNARY
| sizeof '(' typename ')' %prec HYPERUNARY
| alignof unary_expr %prec UNARY
| alignof '(' typename ')' %prec HYPERUNARY
| REALPART cast_expr %prec UNARY
| IMAGPART cast_expr %prec UNARY
;

sizeof:
    SIZEOF
;

alignof:
    ALIGNOF
;

cast_expr:
    unary_expr
    | '(' typename ')' cast_expr %prec UNARY
;

expr_no_commas:
    cast_expr
    | expr_no_commas '+' expr_no_commas
    | expr_no_commas '-' expr_no_commas
    | expr_no_commas '*' expr_no_commas
    | expr_no_commas '/' expr_no_commas
    | expr_no_commas '%' expr_no_commas
    | expr_no_commas LSHIFT expr_no_commas
    | expr_no_commas RSHIFT expr_no_commas
    | expr_no_commas ARITHCOMPARE expr_no_commas
    | expr_no_commas EQCOMPARE expr_no_commas
    | expr_no_commas '&' expr_no_commas
    | expr_no_commas '|' expr_no_commas
    | expr_no_commas '^' expr_no_commas
    | expr_no_commas ANDAND expr_no_commas
    | expr_no_commas OROR expr_no_commas
    | expr_no_commas '?' expr ':' expr_no_commas
    | expr_no_commas '? ':' expr_no_commas
    | expr_no_commas '=' expr_no_commas
    | expr_no_commas ASSIGN expr_no_commas
;

primary:
    IDENTIFIER
    | CONSTANT
    | string
    | VAR_FUNC_NAME
    | '(' typename ')' '{' initlist_maybe_comma '}' %prec UNARY
    | '(' expr ')'
    | '(' error ')'

```

```

| compstmt_primary_start compstmt_nostart ')'
| compstmt_primary_start error ')'
| primary '(' exprlist ')' %prec '.'
| VA_ARG '(' expr_no_commas ',' typename ')'
| CHOOSE_EXPR '(' expr_no_commas ',' expr_no_commas ',' expr_no_commas ')'
| TYPES_COMPATIBLE_P '(' typename ',' typename ')'
| primary '[' expr ']' %prec '.'
| primary '.' identifier
| primary POINTSAT identifier
| primary PLUSPLUS
| primary MINUSMINUS
;

/* Produces a STRING_CST with perhaps more STRING_CSTs chained onto it. */
string:
    STRING
    | string STRING
;

old_style_parm_decls:
    /* empty */
    | datadecls
    | datadecls ELLIPSIS
        /* ... is used here to indicate a varargs function. */
;

/* The following are analogous to lineno_decl, decls and decl
   except that they do not allow nested functions.
   They are used for old-style parm decls. */
lineno_datadecl:
    save_filename save_lineno datadecl
;

datadecls:
    lineno_datadecl
    | errstmt
    | datadecls lineno_datadecl
    | lineno_datadecl errstmt
;

/* We don't allow prefix attributes here because they cause reduce/reduce
   conflicts: we can't know whether we're parsing a function decl with
   attribute suffix, or function defn with attribute prefix on first old
   style parm. */
datadecl:
    declspecs_ts_nosa setspecs initdecls ';'
    | declspecs_nots_nosa setspecs notype_initdecls ';'
    | declspecs_ts_nosa ';'
    | declspecs_nots_nosa ';'
;

/* This combination which saves a lineno before a decl
   is the normal thing to use, rather than decl itself.
   This is to avoid shift/reduce conflicts in contexts
   where statement labels are allowed. */
lineno_decl:
    save_filename save_lineno decl

```

```

;

/* records the type and storage class specs to use for processing
the declarators that follow.
Maintains a stack of outer-level values of current_declspecs,
for the sake of parm declarations nested in function declarators. */
setspecs: /* empty */
;

/* Possibly attributes after a comma, which should reset all_prefix_attributes
to prefix_attributes with these ones chained on the front. */
maybe_resetattrs:
    maybe_attribute
;

decl:
    declspecs_ts setspecs initdecls `;`
    | declspecs_nots setspecs notype_initdecls `;`
    | declspecs_ts setspecs nested_function
    | declspecs_nots setspecs notype_nested_function
    | declspecs `;`
    | extension decl
;

```

/ A list of declaration specifiers. These are:*

- Storage class specifiers (SCSPEC), which for GCC currently include function specifiers ("inline").
- Type specifiers (typespec_*).
- Type qualifiers (TYPE_QUAL).
- Attribute specifier lists (attributes).

These are stored as a TREE_LIST; the head of the list is the last item in the specifier list. Each entry in the list has either a TREE_PURPOSE that is an attribute specifier list, or a TREE_VALUE that is a single other specifier or qualifier; and a TREE_CHAIN that is the rest of the list. TREE_STATIC is set on the list if something other than a storage class specifier or attribute has been seen; this is used to warn for the obsolescent usage of storage class specifiers other than at the start of the list. (Doing this properly would require function specifiers to be handled separately from storage class specifiers.)

The various cases below are classified according to:

- (a) *Whether a storage class specifier is included or not; some places in the grammar disallow storage class specifiers (_sc or _nosc).*
- (b) *Whether a type specifier has been seen; after a type specifier, a typedef name is an identifier to redeclare (_ts or _nots).*
- (c) *Whether the list starts with an attribute; in certain places, the grammar requires specifiers that don't start with an attribute (_sa or _nosa).*
- (d) *Whether the list ends with an attribute (or a specifier such that*

any following attribute would have been parsed as part of that specifier);
this avoids shift-reduce conflicts in the parsing of attributes
(_ea or _noea).

TODO:

(i) Distinguish between function specifiers and storage class specifiers,
at least for the purpose of warnings about obsolescent usage.

(ii) Halve the number of productions here by eliminating the _sc/_nosc
distinction and instead checking where required that storage class
specifiers aren't present. */

/* Declspecs which contain at least one type specifier or typedef name.
(Just `const` or `volatile` is not enough.)
A typedef'd name following these is taken as a name to be declared.
Declspecs have a non-NULL TREE_VALUE, attributes do not. */

```
declspecs_nosc_nots_nosa_noea:  
  TYPE_QUAL  
  | declspecs_nosc_nots_nosa_noea TYPE_QUAL  
  | declspecs_nosc_nots_nosa_ea TYPE_QUAL  
  ;
```

```
declspecs_nosc_nots_nosa_ea:  
  declspecs_nosc_nots_nosa_noea attributes  
  ;
```

```
declspecs_nosc_nots_sa_noea:  
  declspecs_nosc_nots_sa_noea TYPE_QUAL  
  | declspecs_nosc_nots_sa_ea TYPE_QUAL  
  ;
```

```
declspecs_nosc_nots_sa_ea:  
  attributes  
  | declspecs_nosc_nots_sa_noea attributes  
  ;
```

```
declspecs_nosc_ts_nosa_noea:  
  typespec_nonattr  
  | declspecs_nosc_ts_nosa_noea TYPE_QUAL  
  | declspecs_nosc_ts_nosa_ea TYPE_QUAL  
  | declspecs_nosc_ts_nosa_noea typespec_reserved_nonattr  
  | declspecs_nosc_ts_nosa_ea typespec_reserved_nonattr  
  | declspecs_nosc_nots_nosa_noea typespec_nonattr  
  | declspecs_nosc_nots_nosa_ea typespec_nonattr  
  ;
```

```
declspecs_nosc_ts_nosa_ea:  
  typespec_attr  
  | declspecs_nosc_ts_nosa_noea attributes  
  | declspecs_nosc_ts_nosa_noea typespec_reserved_attr  
  | declspecs_nosc_ts_nosa_ea typespec_reserved_attr  
  | declspecs_nosc_nots_nosa_noea typespec_attr  
  | declspecs_nosc_nots_nosa_ea typespec_attr  
  ;
```

```
declspecs_nosc_ts_sa_noea:
```

```

    declspecs_nosc_ts_sa_noea TYPE_QUAL
| declspecs_nosc_ts_sa_ea TYPE_QUAL
| declspecs_nosc_ts_sa_noea typespec_reserved_nonattr
| declspecs_nosc_ts_sa_ea typespec_reserved_nonattr
| declspecs_nosc_nots_sa_noea typespec_nonattr
| declspecs_nosc_nots_sa_ea typespec_nonattr
;

declspecs_nosc_ts_sa_ea:
    declspecs_nosc_ts_sa_noea attributes
| declspecs_nosc_ts_sa_noea typespec_reserved_attr
| declspecs_nosc_ts_sa_ea typespec_reserved_attr
| declspecs_nosc_nots_sa_noea typespec_attr
| declspecs_nosc_nots_sa_ea typespec_attr
;

declspecs_sc_nots_nosa_noea:
    SCSPEC
| declspecs_sc_nots_nosa_noea TYPE_QUAL
| declspecs_sc_nots_nosa_ea TYPE_QUAL
| declspecs_nosc_nots_nosa_noea SCSPEC
| declspecs_nosc_nots_nosa_ea SCSPEC
| declspecs_sc_nots_nosa_noea SCSPEC
| declspecs_sc_nots_nosa_ea SCSPEC
;

declspecs_sc_nots_nosa_ea:
    declspecs_sc_nots_nosa_noea attributes
;

declspecs_sc_nots_sa_noea:
    declspecs_sc_nots_sa_noea TYPE_QUAL
| declspecs_sc_nots_sa_ea TYPE_QUAL
| declspecs_nosc_nots_sa_noea SCSPEC
| declspecs_nosc_nots_sa_ea SCSPEC
| declspecs_sc_nots_sa_noea SCSPEC
| declspecs_sc_nots_sa_ea SCSPEC
;

declspecs_sc_nots_sa_ea:
    declspecs_sc_nots_sa_noea attributes
;

declspecs_sc_ts_nosa_noea:
    declspecs_sc_ts_nosa_noea TYPE_QUAL
| declspecs_sc_ts_nosa_ea TYPE_QUAL
| declspecs_sc_ts_nosa_noea typespec_reserved_nonattr
| declspecs_sc_ts_nosa_ea typespec_reserved_nonattr
| declspecs_sc_nots_nosa_noea typespec_nonattr
| declspecs_sc_nots_nosa_ea typespec_nonattr
| declspecs_nosc_ts_nosa_noea SCSPEC
| declspecs_nosc_ts_nosa_ea SCSPEC
| declspecs_sc_ts_nosa_noea SCSPEC
| declspecs_sc_ts_nosa_ea SCSPEC
;

declspecs_sc_ts_nosa_ea:
    declspecs_sc_ts_nosa_noea attributes

```

```

| declspecs_sc_ts_nosa_noea typespec_reserved_attr
| declspecs_sc_ts_nosa_ea typespec_reserved_attr
| declspecs_sc_nots_nosa_noea typespec_attr
| declspecs_sc_nots_nosa_ea typespec_attr
;

declspecs_sc_ts_sa_noea:
    declspecs_sc_ts_sa_noea TYPE_QUAL
| declspecs_sc_ts_sa_ea TYPE_QUAL
| declspecs_sc_ts_sa_noea typespec_reserved_nonattr
| declspecs_sc_ts_sa_ea typespec_reserved_nonattr
| declspecs_sc_nots_sa_noea typespec_nonattr
| declspecs_sc_nots_sa_ea typespec_nonattr
| declspecs_nosc_ts_sa_noea SCSPEC
| declspecs_nosc_ts_sa_ea SCSPEC
| declspecs_sc_ts_sa_noea SCSPEC
| declspecs_sc_ts_sa_ea SCSPEC
;

declspecs_sc_ts_sa_ea:
    declspecs_sc_ts_sa_noea attributes
| declspecs_sc_ts_sa_noea typespec_reserved_attr
| declspecs_sc_ts_sa_ea typespec_reserved_attr
| declspecs_sc_nots_sa_noea typespec_attr
| declspecs_sc_nots_sa_ea typespec_attr
;

/* Particular useful classes of declspecs. */
declspecs_ts:
    declspecs_nosc_ts_nosa_noea
| declspecs_nosc_ts_nosa_ea
| declspecs_nosc_ts_sa_noea
| declspecs_nosc_ts_sa_ea
| declspecs_sc_ts_nosa_noea
| declspecs_sc_ts_nosa_ea
| declspecs_sc_ts_sa_noea
| declspecs_sc_ts_sa_ea
;

declspecs_nots:
    declspecs_nosc_nots_nosa_noea
| declspecs_nosc_nots_nosa_ea
| declspecs_nosc_nots_sa_noea
| declspecs_nosc_nots_sa_ea
| declspecs_sc_nots_nosa_noea
| declspecs_sc_nots_nosa_ea
| declspecs_sc_nots_sa_noea
| declspecs_sc_nots_sa_ea
;

declspecs_ts_nosa:
    declspecs_nosc_ts_nosa_noea
| declspecs_nosc_ts_nosa_ea
| declspecs_sc_ts_nosa_noea
| declspecs_sc_ts_nosa_ea
;

declspecs_nots_nosa:

```

```

    declspecs_nosc_nots_nosa_noea
  | declspecs_nosc_nots_nosa_ea
  | declspecs_sc_nots_nosa_noea
  | declspecs_sc_nots_nosa_ea
  ;

declspecs_nosc_ts:
    declspecs_nosc_ts_nosa_noea
  | declspecs_nosc_ts_nosa_ea
  | declspecs_nosc_ts_sa_noea
  | declspecs_nosc_ts_sa_ea
  ;

declspecs_nosc_nots:
    declspecs_nosc_nots_nosa_noea
  | declspecs_nosc_nots_nosa_ea
  | declspecs_nosc_nots_sa_noea
  | declspecs_nosc_nots_sa_ea
  ;

declspecs_nosc:
    declspecs_nosc_ts_nosa_noea
  | declspecs_nosc_ts_nosa_ea
  | declspecs_nosc_ts_sa_noea
  | declspecs_nosc_ts_sa_ea
  | declspecs_nosc_nots_nosa_noea
  | declspecs_nosc_nots_nosa_ea
  | declspecs_nosc_nots_sa_noea
  | declspecs_nosc_nots_sa_ea
  ;

declspecs:
    declspecs_nosc_nots_nosa_noea
  | declspecs_nosc_nots_nosa_ea
  | declspecs_nosc_nots_sa_noea
  | declspecs_nosc_nots_sa_ea
  | declspecs_nosc_ts_nosa_noea
  | declspecs_nosc_ts_nosa_ea
  | declspecs_nosc_ts_sa_noea
  | declspecs_nosc_ts_sa_ea
  | declspecs_sc_nots_nosa_noea
  | declspecs_sc_nots_nosa_ea
  | declspecs_sc_nots_sa_noea
  | declspecs_sc_nots_sa_ea
  | declspecs_sc_ts_nosa_noea
  | declspecs_sc_ts_nosa_ea
  | declspecs_sc_ts_sa_noea
  | declspecs_sc_ts_sa_ea
  ;

/* A (possibly empty) sequence of type qualifiers and attributes. */
maybe_type_qual_attrs:
    /* empty */
  | declspecs_nosc_nots
  ;

/* A type specifier (but not a type qualifier).
   Once we have seen one of these in a declaration,
```

if a typedef name appears then it is being redeclared.

The `_reserved` versions start with a reserved word and may appear anywhere in the declaration specifiers; the `_nonreserved` versions may only appear before any other type specifiers, and after that are (if names) being redeclared.

FIXME: should the `_nonreserved` version be restricted to names being redeclared only? The other entries there relate only the GNU extensions and Objective C, and are historically parsed thus, and don't make sense after other type specifiers, but it might be cleaner to count them as `_reserved`.

`_attr` means: specifiers that either end with attributes, or are such that any following attributes would be parsed as part of the specifier.

`_nonattr`: specifiers. **/*

```
typespec_nonattr:
    typespec_reserved_nonattr
    | typespec_nonreserved_nonattr
    ;
```

```
typespec_attr:
    typespec_reserved_attr
    ;
```

```
typespec_reserved_nonattr:
    TYPESPEC
    | structsp_nonattr
    ;
```

```
typespec_reserved_attr:
    structsp_attr
    ;
```

```
typespec_nonreserved_nonattr:
    TYPENAME
    | TYPEOF '(' expr ')'
    | TYPEOF '(' typename ')'
    ;
```

/ typespec_nonreserved_attr does not exist. */*

```
initdecls:
    initdcl
    | initdecls ',' maybe_resetattns initdcl
    ;
```

```
notype_initdecls:
    notype_initdcl
    | notype_initdecls ',' maybe_resetattns notype_initdcl
    ;
```

```
maybeasm:
    /* empty */
    | ASM_KEYWORD '(' string ')'
```

```

;

initdcl:
    declarator maybeasm maybe_attribute '= ' init
/* Note how the declaration of the variable is in effect while its init is parsed! */
    | declarator maybeasm maybe_attribute
;

notype_initdcl:
    notype_declarator maybeasm maybe_attribute '= ' init
/* Note how the declaration of the variable is in effect while its init is parsed! */
    | notype_declarator maybeasm maybe_attribute
;
/* the * rules are dummies to accept the Apollo extended syntax
so that the header files compile. */
maybe_attribute:
    /* empty */
    | attributes
;

attributes:
    attribute
    | attributes attribute
;

attribute:
    ATTRIBUTE '(' '(' attribute_list ')' ')'
;

attribute_list:
    attrib
    | attribute_list ',' attrib
;

attrib:
    /* empty */
    | any_word
    | any_word '(' IDENTIFIER ')'
    | any_word '(' IDENTIFIER ',' nonnull_explist ')'
    | any_word '(' explist ')'
;

/* This still leaves out most reserved keywords,
shouldn't we include them? */

any_word:
    identifier
    | SCSPEC
    | TYPESPEC
    | TYPE_QUAL
;

/* Initializers. `init' is the entry point. */

init:
    expr_no_commas
    | '{' initlist_maybe_comma '}'
    | error

```

```

;

/* `initlist_maybe_comma' is the guts of an initializer in braces. */
initlist_maybe_comma:
    /* empty */
    | initlist1 maybecomma
;

initlist1:
    initelt
    | initlist1 ',' initelt
;

/* `initelt' is a single element of an initializer.
   It may use braces. */
initelt:
    designator_list '=' initval
    | designator initval
    | identifier ':' initval
    | initval
;

initval:
    '{' initlist_maybe_comma '}'
    | expr_no_commas
    | error
;

designator_list:
    designator
    | designator_list designator
;

designator:
    '.' identifier
    /* These are for labeled elements. The syntax for an array element
       initializer conflicts with the syntax for an Objective-C message,
       so don't include these productions in the Objective-C grammar. */
    | '[' expr_no_commas ELLIPSIS expr_no_commas ']'
    | '[' expr_no_commas ']'
;

nested_function:
    declarator old_style_parm_decls
/* This used to use compstmt_or_error.
   That caused a bug with input `f(g) int g {}',
   where the use of YYERROR1 above caused an error
   which then was handled by compstmt_or_error.
   There followed a repeated execution of that same rule,
   which called YYERROR1 again, and so on. */
    save_filename save_lineno compstmt
;

notype_nested_function:
    notype_declarator old_style_parm_decls
/* This used to use compstmt_or_error.
   That caused a bug with input `f(g) int g {}',
   where the use of YYERROR1 above caused an error

```

```

which then was handled by compstmt_or_error.
There followed a repeated execution of that same rule,
which called YYERROR1 again, and so on. */
    save_filename save_lineno compstmt
    ;

```

```

/* Any kind of declarator (thus, all declarators allowed
after an explicit typespec). */

```

```

declarator:
    after_type_declarator
    | notype_declarator
    ;

```

```

/* A declarator that is allowed only after an explicit typespec. */

```

```

after_type_declarator:
    '(' maybe_attribute after_type_declarator ')'
    | after_type_declarator '(' parmlist_or_identifiers %prec '.'
    | after_type_declarator array_declarator %prec '.'
    | '*' maybe_type_qualifiers after_type_declarator %prec UNARY
    | TYPENAME
    ;

```

```

/* Kinds of declarator that can appear in a parameter list
in addition to notype_declarator. This is like after_type_declarator
but does not allow a typedef name in parentheses as an identifier
(because it would conflict with a function with that typedef as arg). */

```

```

parm_declarator:
    parm_declarator_starttypename
    | parm_declarator_nostarttypename
    ;

```

```

parm_declarator_starttypename:
    parm_declarator_starttypename '(' parmlist_or_identifiers %prec '.'
    | parm_declarator_starttypename array_declarator %prec '.'
    | TYPENAME
    ;

```

```

parm_declarator_nostarttypename:
    parm_declarator_nostarttypename '(' parmlist_or_identifiers %prec '.'
    | parm_declarator_nostarttypename array_declarator %prec '.'
    | '*' maybe_type_qualifiers parm_declarator_starttypename %prec UNARY
    | '*' maybe_type_qualifiers parm_declarator_nostarttypename %prec UNARY
    | '(' maybe_attribute parm_declarator_nostarttypename ')'
    ;

```

```

/* A declarator allowed whether or not there has been
an explicit typespec. These cannot redeclare a typedef-name. */

```

```

notype_declarator:
    notype_declarator '(' parmlist_or_identifiers %prec '.'
    | '(' maybe_attribute notype_declarator ')'
    | '*' maybe_type_qualifiers notype_declarator %prec UNARY
    | notype_declarator array_declarator %prec '.'
    | IDENTIFIER
    ;

```

```

struct_head:
    STRUCT
    | STRUCT attributes
    ;

union_head:
    UNION
    | UNION attributes
    ;

enum_head:
    ENUM
    | ENUM attributes
    ;

/* structsp_attr: struct/union/enum specifiers that either
   end with attributes, or are such that any following attributes would
   be parsed as part of the struct/union/enum specifier.

   structsp_nonattr: other struct/union/enum specifiers. */

structsp_attr:
    struct_head identifier '{' component_decl_list '}' maybe_attribute
    | struct_head '{' component_decl_list '}' maybe_attribute
    | union_head identifier '{' component_decl_list '}' maybe_attribute
    | union_head '{' component_decl_list '}' maybe_attribute
    | enum_head identifier '{' enumlist maybecomma_warn '}' maybe_attribute
    | enum_head '{' enumlist maybecomma_warn '}' maybe_attribute
    ;

structsp_nonattr:
    struct_head identifier
    | union_head identifier
    | enum_head identifier
    ;

maybecomma:
    /* empty */
    | ','
    ;

maybecomma_warn:
    /* empty */
    | ','
    ;

component_decl_list:
    component_decl_list2
    | component_decl_list2 component_decl
    ;

component_decl_list2: /* empty */
    | component_decl_list2 component_decl ';'
    | component_decl_list2 ';'
    ;

component_decl:
    declspecs_nosc_ts setspecs components

```

```

    | declspecs_nosc_ts setspecs save_filename save_lineno
    | declspecs_nosc_notypes setspecs components_notype
    | declspecs_nosc_notypes
    | error
    | extension component_decl
    ;

components:
    component_declarator
    | components ',' maybe_resetattrs component_declarator
    ;

components_notype:
    component_notype_declarator
    | components_notype ',' maybe_resetattrs component_notype_declarator
    ;

component_declarator:
    save_filename save_lineno declarator maybe_attribute
    | save_filename save_lineno declarator ':' expr_no_commas maybe_attribute
    | save_filename save_lineno ':' expr_no_commas maybe_attribute
    ;

component_notype_declarator:
    save_filename save_lineno notype_declarator maybe_attribute
    | save_filename save_lineno notype_declarator ':' expr_no_commas maybe_attribute
    | save_filename save_lineno ':' expr_no_commas maybe_attribute
    ;

/* We chain the enumerators in reverse order.
   They are put in forward order where enumlist is used.
   (The order used to be significant, but no longer is so.
   However, we still maintain the order, just to be clean.) */

enumlist:
    enumerator
    | enumlist ',' enumerator
    | error
    ;

enumerator:
    identifier
    | identifier '=' expr_no_commas
    ;

typename:
    declspecs_nosc absdcl
    ;

absdcl: /* an absolute declarator */
    /* empty */
    | absdcl1
    ;

absdcl_maybe_attribute: /* absdcl maybe_attribute, but not just attributes */
    /* empty */
    | absdcl1

```

```

    | absdcl1_noea attributes
    ;

absdcl1: /* a nonempty absolute declarator */
    absdcl1_ea
    | absdcl1_noea
    ;

absdcl1_noea:
    direct_absdcl1
    | '*' maybe_type_qualifiers_attrs absdcl1_noea
    ;

absdcl1_ea:
    '*' maybe_type_qualifiers_attrs
    | '*' maybe_type_qualifiers_attrs absdcl1_ea
    ;

direct_absdcl1:
    '(' maybe_attribute absdcl1 ')'
    | direct_absdcl1 '(' parmlist
    | direct_absdcl1 array_declarator
    | '(' parmlist
    | array_declarator
    ;

/* The [...] part of a declarator for an array type. */

array_declarator:
    '[' expr ']'
    | '[' declspecs_nosc expr ']'
    | '[' ']'
    | '[' declspecs_nosc ']'
    | '[' '*' ']'
    | '[' declspecs_nosc '*' ']'
    | '[' SCSPEC expr ']'
    | '[' SCSPEC declspecs_nosc expr ']'
    | '[' declspecs_nosc SCSPEC expr ']'
    ;

/* A nonempty series of declarations and statements (possibly followed by
some labels) that can form the body of a compound statement.
NOTE: we don't allow labels on declarations; this might seem like a
natural extension, but there would be a conflict between attributes
on the label and prefix attributes on the declaration. */

stmts_and_decls:
    lineno_stmt_decl_or_labels_ending_stmt
    | lineno_stmt_decl_or_labels_ending_decl
    | lineno_stmt_decl_or_labels_ending_label
    | lineno_stmt_decl_or_labels_ending_error
    ;

lineno_stmt_decl_or_labels_ending_stmt:
    lineno_stmt
    | lineno_stmt_decl_or_labels_ending_stmt lineno_stmt
    | lineno_stmt_decl_or_labels_ending_decl lineno_stmt
    | lineno_stmt_decl_or_labels_ending_label lineno_stmt

```

```

        | lineno_stmt_decl_or_labels_ending_error lineno_stmt
        ;

lineno_stmt_decl_or_labels_ending_decl:
    lineno_decl
    | lineno_stmt_decl_or_labels_ending_stmt lineno_decl
    | lineno_stmt_decl_or_labels_ending_decl lineno_decl
    | lineno_stmt_decl_or_labels_ending_error lineno_decl
    ;

lineno_stmt_decl_or_labels_ending_label:
    lineno_label
    | lineno_stmt_decl_or_labels_ending_stmt lineno_label
    | lineno_stmt_decl_or_labels_ending_decl lineno_label
    | lineno_stmt_decl_or_labels_ending_label lineno_label
    | lineno_stmt_decl_or_labels_ending_error lineno_label
    ;

lineno_stmt_decl_or_labels_ending_error:
    errstmt
    | lineno_stmt_decl_or_labels_errstmt
    ;

lineno_stmt_decl_or_labels:
    lineno_stmt_decl_or_labels_ending_stmt
    | lineno_stmt_decl_or_labels_ending_decl
    | lineno_stmt_decl_or_labels_ending_label
    | lineno_stmt_decl_or_labels_ending_error
    ;

errstmt: error `;`
    ;

pushlevel: /* empty */
    ;

poplevel: /* empty */
    ;

/* Start and end blocks created for the new scopes of C99. */
c99_block_start: /* empty */
    ;

/* Productions using c99_block_start and c99_block_end will need to do what's
   in compstmt: RECHAIN_STMTS ($1, COMPOUND_BODY ($1)); $$ = $2; where
   $1 is the value of c99_block_start and $2 of c99_block_end. */
c99_block_end: /* empty */
    ;

/* Read zero or more forward-declarations for labels
   that nested functions can jump to. */
maybe_label_decls:
    /* empty */
    | label_decls
    ;

label_decls:
    label_decl

```

```

        | label_decls label_decl
        ;

label_decl:
    LABEL identifiers_or_typenames ';'
    ;

/* This is the body of a function definition.
   It causes syntax errors to ignore to the next openbrace. */
compstmt_or_error:
    compstmt
    | error compstmt
    ;

compstmt_start: '{'
    ;

compstmt_nostart: '}'
    | pushlevel maybe_label_decls compstmt_contents_nonempty '}' poplevel
    ;

compstmt_contents_nonempty:
    stmts_and_decls
    | error
    ;

compstmt_primary_start:
    '(' '{'
    ;

compstmt: compstmt_start compstmt_nostart
    ;

/* Value is number of statements counted as of the closeparen. */
simple_if:
    if_prefix c99_block_lineno_labeled_stmt
/* Make sure c_expand_end_cond is run once
   for each call to c_expand_start_cond.
   Otherwise a crash is likely. */
    | if_prefix error
    ;

if_prefix:
    /* We must build the IF_STMT node before parsing its
       condition so that STMT_LINENO refers to the line
       containing the "if", and not the line containing
       the close-parenthesis.

       c_begin_if_stmt returns the IF_STMT node, which
       we later pass to c_expand_start_cond to fill
       in the condition and other tidbits. */
    IF '(' expr ')'
    ;

/* This is a subroutine of stmt.
   It is used twice, once for valid DO statements
   and once for catching errors in parsing the end test. */
do_stmt_start:

```

```

        DO c99_block_lineno_labeled_stmt WHILE
    ;

/* The forced readahead in here is because we might be at the end of a
   line, and the line and file won't be bumped until yylex absorbs the
   first token on the next line. */
save_filename:
    ;

save_lineno:
    ;

lineno_labeled_stmt:
    lineno_stmt
    | lineno_label lineno_labeled_stmt
    ;

/* Like lineno_labeled_stmt, but a block in C99. */
c99_block_lineno_labeled_stmt:
    c99_block_start lineno_labeled_stmt c99_block_end
    ;

lineno_stmt:
    save_filename save_lineno stmt
    ;

lineno_label:
    save_filename save_lineno label
    ;

select_or_iter_stmt:
    simple_if ELSE c99_block_lineno_labeled_stmt
    | simple_if %prec IF
/* Make sure c_expand_end_cond is run once
   for each call to c_expand_start_cond.
   Otherwise a crash is likely. */
    | simple_if ELSE error
/* We must build the WHILE_STMT node before parsing its
   condition so that STMT_LINENO refers to the line
   containing the "while", and not the line containing
   the close-parenthesis.

   c_begin_while_stmt returns the WHILE_STMT node, which
   we later pass to c_finish_while_stmt_cond to fill
   in the condition and other tidbits. */
    | WHILE '(' expr ')' c99_block_lineno_labeled_stmt
    | do_stmt_start '(' expr ')' ';'
    | do_stmt_start error
    | FOR '(' for_init_stmt xexpr ';' xexpr ')' c99_block_lineno_labeled_stmt
    | SWITCH '(' expr ')' c99_block_lineno_labeled_stmt
    ;

for_init_stmt:
    xexpr ';'
    | decl
    ;

/* Parse a single real statement, not including any labels. */

```

```

stmt:
    compstmt
    | expr ';'
    | c99_block_start select_or_iter_stmt c99_block_end
    | BREAK ';'
    | CONTINUE ';'
    | RETURN ';'
    | RETURN expr ';'
    | ASM_KEYWORD maybe_type_qual '(' expr ')' ';'
    /* This is the case with just output operands. */
    | ASM_KEYWORD maybe_type_qual '(' expr ':' asm_operands ')' ';'
    /* This is the case with input operands as well. */
    | ASM_KEYWORD maybe_type_qual '(' expr ':' asm_operands ':' asm_operands ')' ';'
    /* This is the case with clobbered registers as well. */
    | ASM_KEYWORD maybe_type_qual '(' expr ':' asm_operands ':' asm_operands ':' asm_clobbers ')' ';'
    | GOTO identifier ';'
    | GOTO '*' expr ';'
    | ';'
    ;

```

/ Any kind of label, including jump labels and case labels.
ANSI C accepts labels only before statements, but we allow them
also at the end of a compound statement. */*

```

label:
    CASE expr_no_commas ':'
    | CASE expr_no_commas ELLIPSIS expr_no_commas ':'
    | DEFAULT ':'
    | identifier save_filename save_lineno ':' maybe_attribute
    ;

```

/ Either a type-qualifier or nothing. First thing in an `asm` statement. */*

```

maybe_type_qual:
    /* empty */
    | TYPE_QUAL
    ;

```

```

xexpr:
    /* empty */
    | expr
    ;

```

/ These are the operands other than the first string and colon
in asm ("addextend %2,%1": "=dm" (x), "0" (y), "g" (*x)) */*

```

asm_operands: /* empty */
    | nonnull_asm_operands
    ;

```

```

nonnull_asm_operands:
    asm_operand
    | nonnull_asm_operands ',' asm_operand
    ;

```

```

asm_operand:
    STRING '(' expr ')'
    | '[' identifier ']' STRING '(' expr ')'
    ;

```

```

asm_clobbers:
    string
    | asm_clobbers ' , ' string
    ;

/* This is what appears inside the parens in a function declarator.
   Its value is a list of ... TYPE nodes. Attributes must appear here
   to avoid a conflict with their appearance after an open parenthesis
   in an abstract declarator, as in
   "void bar (int ( __ attribute__ (( __ mode__ (SI))) int foo));". */
parmlist:
    maybe_attribute parmlist_1
    ;

parmlist_1:
    parmlist_2 ' ) '
    | parms ' ; ' maybe_attribute parmlist_1
    | error ' ) '
    ;

/* This is what appears inside the parens in a function declarator.
   Its value is represented in the format that grokdeclarator expects. */
parmlist_2: /* empty */
    | ELLIPSIS
    | parms
    | parms ' , ' ELLIPSIS
    ;

parms:
    firstparm
    | parms ' , ' parm
    ;

/* A single parameter declaration or parameter type name,
   as found in a parmlist. */
parm:
    declspecs_ts setspecs parm_declarator maybe_attribute
    | declspecs_ts setspecs notype_declarator maybe_attribute
    | declspecs_ts setspecs absdcl_maybe_attribute
    | declspecs_not_s setspecs notype_declarator maybe_attribute

    | declspecs_not_s setspecs absdcl_maybe_attribute
    ;

/* The first parm, which must suck attributes from off the top of the parser
   stack. */
firstparm:
    declspecs_ts_nosa setspecs_fp parm_declarator maybe_attribute
    | declspecs_ts_nosa setspecs_fp notype_declarator maybe_attribute
    | declspecs_ts_nosa setspecs_fp absdcl_maybe_attribute
    | declspecs_not_s_nosa setspecs_fp notype_declarator maybe_attribute

    | declspecs_not_s_nosa setspecs_fp absdcl_maybe_attribute
    ;

setspecs_fp:
    setspecs
    ;

```

```

/* This is used in a function definition
   where either a parmlist or an identifier list is ok.
   Its value is a list of ..._TYPE nodes or a list of identifiers. */
parmlist_or_identifiers:
    maybe_attribute parmlist_or_identifiers_1
    ;

parmlist_or_identifiers_1:
    parmlist_1
    | identifiers `)`
    ;

/* A nonempty list of identifiers. */
identifiers:
    IDENTIFIER
    | identifiers `,` IDENTIFIER
    ;

/* A nonempty list of identifiers, including typenames. */
identifiers_or_typenames:
    identifier
    | identifiers_or_typenames `,` identifier
    ;

extension:
    EXTENSION
    ;

```