

Annex A

(informative)

Language syntax summary

- 1 NOTE The notation is described in 6.1.

A.1 Lexical grammar

A.1.1 Lexical elements

(6.4) *token*:

keyword
identifier
constant
string-literal
punctuator

(6.4) *preprocessing-token*:

header-name
identifier
pp-number
character-constant
string-literal
punctuator

each non-white-space character that cannot be one of the above

A.1.2 Keywords

(6.4.1) *keyword*: one of

auto	enum	restrict	unsigned
break	extern	return	void
case	float	short	volatile
char	for	signed	while
const	goto	sizeof	_Bool
continue	if	static	_Complex
default	inline	struct	_Imaginary
do	int	switch	
double	long	typedef	
else	register	union	

A.1.3 Identifiers

(6.4.2.1) *identifier*:

identifier-nondigit
identifier identifier-nondigit
identifier digit

(6.4.2.1) *identifier-nondigit*:

nondigit
universal-character-name
 other implementation-defined characters

(6.4.2.1) *nondigit*: one of

—	a	b	c	d	e	f	g	h	i	j	k	l	m
	n	o	p	q	r	s	t	u	v	w	x	y	z
	A	B	C	D	E	F	G	H	I	J	K	L	M
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

(6.4.2.1) *digit*: one of

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

A.1.4 Universal character names

(6.4.3) *universal-character-name*:

\u hex-quad
\U hex-quad hex-quad

(6.4.3) *hex-quad*:

hexadecimal-digit hexadecimal-digit
hexadecimal-digit hexadecimal-digit

A.1.5 Constants

(6.4.4) *constant*:

integer-constant
floating-constant
enumeration-constant
character-constant

(6.4.4.1) *integer-constant*:

decimal-constant integer-suffix_{opt}
octal-constant integer-suffix_{opt}
hexadecimal-constant integer-suffix_{opt}

(6.4.4.1) *decimal-constant*:

nonzero-digit
decimal-constant digit

(6.4.4.1) *octal-constant*:

0
octal-constant octal-digit

(6.4.4.1) *hexadecimal-constant*:

hexadecimal-prefix hexadecimal-digit
hexadecimal-constant hexadecimal-digit

(6.4.4.1) *hexadecimal-prefix*: one of

0x 0X

(6.4.4.1) *nonzero-digit*: one of

1 2 3 4 5 6 7 8 9

(6.4.4.1) *octal-digit*: one of

0 1 2 3 4 5 6 7

(6.4.4.1) *hexadecimal-digit*: one of

0 1 2 3 4 5 6 7 8 9
a b c d e f
A B C D E F

(6.4.4.1) *integer-suffix*:

unsigned-suffix long-suffix_{opt}
unsigned-suffix long-long-suffix
long-suffix unsigned-suffix_{opt}
long-long-suffix unsigned-suffix_{opt}

(6.4.4.1) *unsigned-suffix*: one of

u U

(6.4.4.1) *long-suffix*: one of

l L

(6.4.4.1) *long-long-suffix*: one of

ll LL

(6.4.4.2) *floating-constant*:

decimal-floating-constant
hexadecimal-floating-constant

(6.4.4.2) *decimal-floating-constant*:

fractional-constant exponent-part_{opt} floating-suffix_{opt}
digit-sequence exponent-part floating-suffix_{opt}

(6.4.4.2) *hexadecimal-floating-constant*:

hexadecimal-prefix hexadecimal-fractional-constant
binary-exponent-part floating-suffix_{opt}
hexadecimal-prefix hexadecimal-digit-sequence
binary-exponent-part floating-suffix_{opt}

(6.4.4.2) *fractional-constant*:

digit-sequence_{opt} . digit-sequence
digit-sequence .

(6.4.4.2) *exponent-part*:

e *sign_{opt} digit-sequence*
E *sign_{opt} digit-sequence*

(6.4.4.2) *sign*: one of

+ **-**

(6.4.4.2) *digit-sequence*:

digit
digit-sequence digit

(6.4.4.2) *hexadecimal-fractional-constant*:

hexadecimal-digit-sequence_{opt} .
hexadecimal-digit-sequence
hexadecimal-digit-sequence .

(6.4.4.2) *binary-exponent-part*:

p *sign_{opt} digit-sequence*
P *sign_{opt} digit-sequence*

(6.4.4.2) *hexadecimal-digit-sequence*:

hexadecimal-digit
hexadecimal-digit-sequence hexadecimal-digit

(6.4.4.2) *floating-suffix*: one of

f **l** **F** **L**

(6.4.4.3) *enumeration-constant*:

identifier

(6.4.4.4) *character-constant*:

' *c-char-sequence* **'**
L' *c-char-sequence* **'**

(6.4.4.4) *c-char-sequence*:

c-char
c-char-sequence c-char

(6.4.4.4) *c-char*:

any member of the source character set except
the single-quote ' , backslash \, or new-line character
escape-sequence

(6.4.4.4) *escape-sequence*:

simple-escape-sequence
octal-escape-sequence
hexadecimal-escape-sequence
universal-character-name

(6.4.4.4) *simple-escape-sequence*: one of

\ ' \ " \ ? \ \
\ a \ b \ f \ n \ r \ t \ v

(6.4.4.4) *octal-escape-sequence*:

\ *octal-digit*
\ *octal-digit octal-digit*
\ *octal-digit octal-digit octal-digit*

(6.4.4.4) *hexadecimal-escape-sequence*:

\ **x** *hexadecimal-digit*
hexadecimal-escape-sequence hexadecimal-digit

A.1.6 String literals

(6.4.5) *string-literal*:

" *s-char-sequence_{opt}* "
L" *s-char-sequence_{opt}* "

(6.4.5) *s-char-sequence*:

s-char
s-char-sequence s-char

(6.4.5) *s-char*:

any member of the source character set except
the double-quote " , backslash \, or new-line character
escape-sequence

A.1.7 Punctuators

(6.4.6) *punctuator*: one of

```
[ ] ( ) { } . ->
++ -- & * + - ~ !
/ % << >> < > <= >= == != ^ | && ||
? : ; ...
= *= /= %= += -= <<= >>= &= ^= |=
, # ##
<: :> <% %> %: %::%
```

A.1.8 Header names

(6.4.7) *header-name*:

```
< h-char-sequence >
" q-char-sequence "
```

(6.4.7) *h-char-sequence*:

```
h-char
h-char-sequence h-char
```

(6.4.7) *h-char*:

any member of the source character set except
the new-line character and >

(6.4.7) *q-char-sequence*:

```
q-char
q-char-sequence q-char
```

(6.4.7) *q-char*:

any member of the source character set except
the new-line character and "

A.1.9 Preprocessing numbers

(6.4.8) *pp-number*:

```
digit
. digit
pp-number digit
pp-number identifier-nondigit
pp-number e sign
pp-number E sign
pp-number p sign
pp-number P sign
pp-number .
```

A.2 Phrase structure grammar

A.2.1 Expressions

(6.5.1) *primary-expression*:

identifier
constant
string-literal
 (*expression*)

(6.5.2) *postfix-expression*:

primary-expression
postfix-expression [*expression*]
postfix-expression (*argument-expression-list*_{opt})
postfix-expression . *identifier*
postfix-expression -> *identifier*
postfix-expression ++
postfix-expression --
 (*type-name*) { *initializer-list* }
 (*type-name*) { *initializer-list* , }

(6.5.2) *argument-expression-list*:

assignment-expression
argument-expression-list , *assignment-expression*

(6.5.3) *unary-expression*:

postfix-expression
 ++ *unary-expression*
 -- *unary-expression*
unary-operator *cast-expression*
sizeof *unary-expression*
sizeof (*type-name*)

(6.5.3) *unary-operator*: one of

& * + - ~ !

(6.5.4) *cast-expression*:

unary-expression
 (*type-name*) *cast-expression*

(6.5.5) *multiplicative-expression*:

cast-expression
multiplicative-expression * *cast-expression*
multiplicative-expression / *cast-expression*
multiplicative-expression % *cast-expression*

(6.5.6) *additive-expression:**multiplicative-expression**additive-expression* **+** *multiplicative-expression**additive-expression* **-** *multiplicative-expression*(6.5.7) *shift-expression:**additive-expression**shift-expression* **<<** *additive-expression**shift-expression* **>>** *additive-expression*(6.5.8) *relational-expression:**shift-expression**relational-expression* **<** *shift-expression**relational-expression* **>** *shift-expression**relational-expression* **<=** *shift-expression**relational-expression* **>=** *shift-expression*(6.5.9) *equality-expression:**relational-expression**equality-expression* **==** *relational-expression**equality-expression* **!=** *relational-expression*(6.5.10) *AND-expression:**equality-expression**AND-expression* **&** *equality-expression*(6.5.11) *exclusive-OR-expression:**AND-expression**exclusive-OR-expression* **^** *AND-expression*(6.5.12) *inclusive-OR-expression:**exclusive-OR-expression**inclusive-OR-expression* **|** *exclusive-OR-expression*(6.5.13) *logical-AND-expression:**inclusive-OR-expression**logical-AND-expression* **&&** *inclusive-OR-expression*(6.5.14) *logical-OR-expression:**logical-AND-expression**logical-OR-expression* **||** *logical-AND-expression*(6.5.15) *conditional-expression:**logical-OR-expression**logical-OR-expression* **?** *expression* **:** *conditional-expression*

(6.5.16) *assignment-expression*:

conditional-expression

unary-expression assignment-operator assignment-expression

(6.5.16) *assignment-operator*: one of

= *= /= %= += -= <<= >>= &= ^= |=

(6.5.17) *expression*:

assignment-expression

expression , assignment-expression

(6.6) *constant-expression*:

conditional-expression

A.2.2 Declarations

(6.7) *declaration*:

declaration-specifiers init-declarator-list_{opt} ;

(6.7) *declaration-specifiers*:

storage-class-specifier declaration-specifiers_{opt}

type-specifier declaration-specifiers_{opt}

type-qualifier declaration-specifiers_{opt}

function-specifier declaration-specifiers_{opt}

(6.7) *init-declarator-list*:

init-declarator

init-declarator-list , init-declarator

(6.7) *init-declarator*:

declarator

declarator = initializer

(6.7.1) *storage-class-specifier*:

typedef

extern

static

auto

register

(6.7.2) *type-specifier*:

void
char
short
int
long
float
double
signed
unsigned
_Bool
_Complex
_Imaginary
struct-or-union-specifier
enum-specifier
typedef-name

(6.7.2.1) *struct-or-union-specifier*:

struct-or-union identifier_{opt} { struct-declaration-list }
struct-or-union identifier

(6.7.2.1) *struct-or-union*:

struct
union

(6.7.2.1) *struct-declaration-list*:

struct-declaration
struct-declaration-list struct-declaration

(6.7.2.1) *struct-declaration*:

specifier-qualifier-list struct-declarator-list ;

(6.7.2.1) *specifier-qualifier-list*:

type-specifier specifier-qualifier-list_{opt}
type-qualifier specifier-qualifier-list_{opt}

(6.7.2.1) *struct-declarator-list*:

struct-declarator
struct-declarator-list , struct-declarator

(6.7.2.1) *struct-declarator*:

declarator
declarator_{opt} : constant-expression

(6.7.2.2) *enum-specifier*:

enum *identifier*_{opt} { *enumerator-list* }
enum *identifier*_{opt} { *enumerator-list* , }
enum *identifier*

(6.7.2.2) *enumerator-list*:

enumerator
enumerator-list , *enumerator*

(6.7.2.2) *enumerator*:

enumeration-constant
enumeration-constant = *constant-expression*

(6.7.3) *type-qualifier*:

const
restrict
volatile

(6.7.4) *function-specifier*:

inline

(6.7.5) *declarator*:

*pointer*_{opt} *direct-declarator*

(6.7.5) *direct-declarator*:

identifier
(*declarator*)
direct-declarator [*type-qualifier-list*_{opt} *assignment-expression*_{opt}]
direct-declarator [**static** *type-qualifier-list*_{opt} *assignment-expression*]
direct-declarator [*type-qualifier-list* **static** *assignment-expression*]
direct-declarator [*type-qualifier-list*_{opt} *]
direct-declarator (*parameter-type-list*)
direct-declarator (*identifier-list*_{opt})

(6.7.5) *pointer*:

* *type-qualifier-list*_{opt}
* *type-qualifier-list*_{opt} *pointer*

(6.7.5) *type-qualifier-list*:

type-qualifier
type-qualifier-list *type-qualifier*

(6.7.5) *parameter-type-list*:

parameter-list
parameter-list , ...

(6.7.5) *parameter-list*:

parameter-declaration
parameter-list , *parameter-declaration*

(6.7.5) *parameter-declaration*:

declaration-specifiers declarator
declaration-specifiers abstract-declarator_{opt}

(6.7.5) *identifier-list*:

identifier
identifier-list , *identifier*

(6.7.6) *type-name*:

specifier-qualifier-list abstract-declarator_{opt}

(6.7.6) *abstract-declarator*:

pointer
pointer_{opt} direct-abstract-declarator

(6.7.6) *direct-abstract-declarator*:

(*abstract-declarator*)
direct-abstract-declarator_{opt} [assignment-expression_{opt}]
*direct-abstract-declarator_{opt} [*]*
direct-abstract-declarator_{opt} (parameter-type-list_{opt})

(6.7.7) *typedef-name*:

identifier

(6.7.8) *initializer*:

assignment-expression
{ *initializer-list* }
{ *initializer-list* , }

(6.7.8) *initializer-list*:

designation_{opt} initializer
initializer-list , *designation_{opt} initializer*

(6.7.8) *designation*:

designator-list =

(6.7.8) *designator-list*:

designator
designator-list designator

(6.7.8) *designator*:

[*constant-expression*]
 . *identifier*

A.2.3 Statements

(6.8) *statement*:

labeled-statement
compound-statement
expression-statement
selection-statement
iteration-statement
jump-statement

(6.8.1) *labeled-statement*:

identifier : *statement*
case *constant-expression* : *statement*
default : *statement*

(6.8.2) *compound-statement*:

{ *block-item-list*_{opt} }

(6.8.2) *block-item-list*:

block-item
block-item-list *block-item*

(6.8.2) *block-item*:

declaration
statement

(6.8.3) *expression-statement*:

*expression*_{opt} ;

(6.8.4) *selection-statement*:

if (*expression*) *statement*
if (*expression*) *statement* **else** *statement*
switch (*expression*) *statement*

(6.8.5) *iteration-statement*:

while (*expression*) *statement*
do *statement* **while** (*expression*) ;
for (*expression*_{opt} ; *expression*_{opt} ; *expression*_{opt}) *statement*
for (*declaration* *expression*_{opt} ; *expression*_{opt}) *statement*

(6.8.6) *jump-statement*:

```
goto identifier ;
continue ;
break ;
return expressionopt ;
```

A.2.4 External definitions

(6.9) *translation-unit*:

```
external-declaration
translation-unit external-declaration
```

(6.9) *external-declaration*:

```
function-definition
declaration
```

(6.9.1) *function-definition*:

```
declaration-specifiers declarator declaration-listopt compound-statement
```

(6.9.1) *declaration-list*:

```
declaration
declaration-list declaration
```

A.3 Preprocessing directives

(6.10) *preprocessing-file*:

```
groupopt
```

(6.10) *group*:

```
group-part
group group-part
```

(6.10) *group-part*:

```
if-section
control-line
text-line
# non-directive
```

(6.10) *if-section*:

```
if-group elif-groupsopt else-groupopt endif-line
```

(6.10) *if-group*:

```
# if      constant-expression new-line groupopt
# ifdef  identifier new-line groupopt
# ifndef  identifier new-line groupopt
```

(6.10) *elif-groups*:

elif-group
elif-groups elif-group

(6.10) *elif-group*:

elif *constant-expression new-line group_{opt}*

(6.10) *else-group*:

else *new-line group_{opt}*

(6.10) *endif-line*:

endif *new-line*

(6.10) *control-line*:

include *pp-tokens new-line*
define *identifier replacement-list new-line*
define *identifier lparen identifier-list_{opt})*
replacement-list new-line
define *identifier lparen ...) replacement-list new-line*
define *identifier lparen identifier-list , ...)*
replacement-list new-line
undef *identifier new-line*
line *pp-tokens new-line*
error *pp-tokens_{opt} new-line*
pragma *pp-tokens_{opt} new-line*
*new-line*

(6.10) *text-line*:

pp-tokens_{opt} new-line

(6.10) *non-directive*:

pp-tokens new-line

(6.10) *lparen*:

a (character not immediately preceded by white-space

(6.10) *replacement-list*:

pp-tokens_{opt}

(6.10) *pp-tokens*:

preprocessing-token
pp-tokens preprocessing-token

(6.10) *new-line*:

the new-line character