1 External Definitions

translation-unit:
  external-declaration
  translation-unit external-declaration

external-declaration:
  declaration

2 Declarations

declaration-list:
  declaration
  declaration-list declaration

declaration:
  inlined-declaration
  `(`*identifier type-expression initializer*$_{opt}$`)`
  `(`*function-identifier* `(fn` *function-type-list*`)`
    `[:attr` *function-attribute*`]`$_{opt}$*register-declarator*$_{opt}$
    block-item-list$_{opt}$`)`
  `(`*struct-or-union-specifier struct-declaration-list*$_{opt}$`)`
  `(`*enum-specifier enumerator-list* `)`

inlined-declaration-list:
  inlined-declaration
  inlined-declaration-list declaration

inlined-declaration:
  `(`*storage-class-specifier identifier type-expression*
    initializer$_{opt}$`)`
  `(`*storage-class-specifier function-identifier*
    `(fn` *function-type-list*`)`
    `[:attr` *function-attribute*`]`$_{opt}$*register-declarator*$_{opt}$
    block-item-list$_{opt}$`)`
  `(`*def-or-decl struct-or-union-specifier*
    struct-declaration-list$_{opt}$`)`
  `(def` *enum-specifier enumerator-list* `)`
  `(`*compound-storage-class-specifier type-expression*
    init-declarator-list `)`
  `(deftype` *identifier type-expression* `)`
  `(deftype` *identifier struct-or-union*
    struct-declaration-list$_{opt}$`)`
  `(deftype` *identifier* `enum` *enumerator-list* `)`

function-identifier:
  identifier
  `(`*identifier-list*`)`

def-or-decl:
  `def`
  `decl`

init-declarator-list:
  init-declarator
  init-declarator-list init-declarator

init-declarator:
  identifier
  `(`*identifier initializer*`)`

storage-class-specifier: one of
  `def decl extern extern-def`
  `extern-decl static static-def`
  `auto auto-def register register-def`

compound-storage-class-specifier: one of
  `defs extern-defs static-defs`
`auto-defs register-defs`

function-attribute:
  `inline`

register-declarator:
  `(register` *identifier-list*`)`

struct-declaration-list:
  struct-declaration
  struct-declaration-list struct-declaration

struct-declaration:
  declaration `[:bit` *expression*`]`$_{opt}$

enumerator-list:
  enumerator
  enumerator-list enumerator

enumerator:
  enumeration-constant

(*enumeration-constant expression*)

enmuration-constant:
  identifier

identifier-list:
  identifier
  identifier-list identifier

designator:
  (`aref-this` *expression-list*)
  (`fref-this` *identifier-list*)
  (`aref` *designator expression-list*)
  (`fref` *designator identifier-list*)

designated-initializer:
  initializer
  (*designator initializer*)

initializer-list:
  designated-initializer
  initializer-list designated-initializer

compound-initializer:
  (`array` *initializer-list*)
  (`struct` *initializer-list*)

initializer:
  expression
  compound-initializer

3 Type-expressions

type-expression:
  type-specifier
  (*type-qualifier-list type-expression*)
  (`array` *type-expression array-subscription-list*$_{opt}$)
  (`ptr` *type-expression*)
  (`fn` *function-type-list*)

function-type-list:
  type-expression-list `va-arg`$_{opt}$

type-expression-list
  type-expression
  type-expression-list type-expression

type-specifier: one of
  `void`
  `char signed-char unsigned-char`
  `short signed-short unsigned-short`
  `int signed-int unsigned-int`
  `long signed-long unsigned-long`
  `long-long signed-long-long`
  `unsigned-long-long`
  `float double long-double`
  struct-or-union-specifier
  enum-specifier
  typedef-name

array-subscription-list:
  expression-list

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

struct-or-union:
  `struct`
  `union`

enum-specifier:
  (`enum` *identifier*)

type-qualifier-list:
  type-qualifier
  type-qualifier-list type-qualifier

type-qualifier:
  `const`
  `restrict`
  `volatile`

typedef-name:
  identifier

4 Statements

statement:
  compound-statement
  expression-statement
  selection-statement
  iteration-statement
  jump-statement
  labeled-statement
  `()`

compound-statement:
  `(begin` *block-item-list*$_{opt}$ `)`
  `(let (`*declaration-list*$_{opt}$ `)` *block-item-list*$_{opt}$ `)`

block-item-list:
  block-item
  block-item-list block-item

block-item:
  inlined-declaration
  statement

labeled-statement:
  `(label` *identifier statement* `)`
  `(case` *expression* `)`
  `(default)`

expression-statement:
  expression

selection-statement:
  `(if` *expression statement statement*$_{opt}$ `)`
  `(switch` *expression block-item-list*$_{opt}$ `)`

iteration-statement:
  `(while` *expression block-item-list*$_{opt}$ `)`
  `(do-while` *expression block-item-list*$_{opt}$ `)`
  `(for (`*expression-list*$_{opt}$ *expression expression* `)`
    block-item-list$_{opt}$ `)`
  `(for (`*inlined-declaration-list*$_{opt}$
      expression expression `)`
    block-item-list$_{opt}$ `)`
  `(loop` *block-item-list*$_{opt}$ `)`

jump-statement:
  `(goto` *identifier* `)`
  `(continue)`
  `(break)`
  `(return` *expression*$_{opt}$ `)`

5 Expressions

expression:
  identifier
  constant
  string-literal
  compound-literal
  `(`*expression-list*`)`
  `(aref` *expression-list* `)`
  `(fref` *expression field-identifier-list* `)`
  `(inc` *expression* `)`
  `(dec` *expression* `)`
  `(++` *experssion* `)`
  `(--` *expression* `)`
  `(`*unary-operator expression* `)`
  `(sizeof` *expression* `)`
  `(sizeof` *type-expression* `)`
  `(cast` *type-expression expression* `)`
  `(`*operator expression-list* `)`
  `(`*comparator expression expression* `)`
  `(if-exp` *expression expression expression* `)`
  `(`*assignment-operator expression expression* `)`
  `(exps` *expression-list* `)`

compound-literal:
  `(init` *type-expression compound-initializer*`)`

expression-list:
  expression
  expression-list expression

field-identifier-list:
  field-identifier
  field-identifier-list field-identifier

field-identifier:
  identifier
  `->` *identifier*

operator: one of

```
  * / % + - << >>
  bit-xor bit-and bit-or and or
```

comparator: one of

```
  < > <= >= == !=
```

assignment-operator: one of

```
  = *= /= %= += -= <<= >>= bit-and=
  bit-xor= bit-or=
```

unary-operator: one of

```
  ptr mref bit-not not
```