#-----------------------------------------------------------------

# pycparser: \_c\_ast.cfg

#

# Defines the AST Node classes used in pycparser.

#

# Each entry is a Node sub-class name, listing the attributes

# and child nodes of the class:

# <name>\* - a child node

# <name>\*\* - a sequence of child nodes

# <name> - an attribute

#

# Eli Bendersky [http://eli.thegreenplace.net]

# License: BSD

#-----------------------------------------------------------------

# ArrayDecl is a nested declaration of an array with the given type.

# dim: the dimension (for example, constant 42)

# dim\_quals: list of dimension qualifiers, to support C99's allowing 'const'

# and 'static' within the array dimension in function declarations.

ArrayDecl: [type\*, dim\*, dim\_quals]

ArrayRef: [name\*, subscript\*]

# op: =, +=, /= etc.

#

Assignment: [op, lvalue\*, rvalue\*]

BinaryOp: [op, left\*, right\*]

Break: []

Case: [expr\*, stmts\*\*]

Cast: [to\_type\*, expr\*]

# Compound statement in C99 is a list of block items (declarations or

# statements).

#

Compound: [block\_items\*\*]

# Compound literal (anonymous aggregate) for C99.

# (type-name) {initializer\_list}

# type: the typename

# init: InitList for the initializer list

#

CompoundLiteral: [type\*, init\*]

# type: int, char, float, etc. see CLexer for constant token types

#

Constant: [type, value]

Continue: []

# name: the variable being declared

# quals: list of qualifiers (const, volatile)

# funcspec: list function specifiers (i.e. inline in C99)

# storage: list of storage specifiers (extern, register, etc.)

# type: declaration type (probably nested with all the modifiers)

# init: initialization value, or None

# bitsize: bit field size, or None

#

Decl: [name, quals, storage, funcspec, type\*, init\*, bitsize\*]

DeclList: [decls\*\*]

Default: [stmts\*\*]

DoWhile: [cond\*, stmt\*]

# Represents the ellipsis (...) parameter in a function

# declaration

#

EllipsisParam: []

# An empty statement (a semicolon ';' on its own)

#

EmptyStatement: []

# Enumeration type specifier

# name: an optional ID

# values: an EnumeratorList

#

Enum: [name, values\*]

# A name/value pair for enumeration values

#

Enumerator: [name, value\*]

# A list of enumerators

#

EnumeratorList: [enumerators\*\*]

# A list of expressions separated by the comma operator.

#

ExprList: [exprs\*\*]

# This is the top of the AST, representing a single C file (a

# translation unit in K&R jargon). It contains a list of

# "external-declaration"s, which is either declarations (Decl),

# Typedef or function definitions (FuncDef).

#

FileAST: [ext\*\*]

# for (init; cond; next) stmt

#

For: [init\*, cond\*, next\*, stmt\*]

# name: Id

# args: ExprList

#

FuncCall: [name\*, args\*]

# type <decl>(args)

#

FuncDecl: [args\*, type\*]

# Function definition: a declarator for the function name and

# a body, which is a compound statement.

# There's an optional list of parameter declarations for old

# K&R-style definitions

#

FuncDef: [decl\*, param\_decls\*\*, body\*]

Goto: [name]

ID: [name]

# Holder for types that are a simple identifier (e.g. the built

# ins void, char etc. and typedef-defined types)

#

IdentifierType: [names]

If: [cond\*, iftrue\*, iffalse\*]

# An initialization list used for compound literals.

#

InitList: [exprs\*\*]

Label: [name, stmt\*]

# A named initializer for C99.

# The name of a NamedInitializer is a sequence of Nodes, because

# names can be hierarchical and contain constant expressions.

#

NamedInitializer: [name\*\*, expr\*]

# a list of comma separated function parameter declarations

#

ParamList: [params\*\*]

PtrDecl: [quals, type\*]

Return: [expr\*]

# name: struct tag name

# decls: declaration of members

#

Struct: [name, decls\*\*]

# type: . or ->

# name.field or name->field

#

StructRef: [name\*, type, field\*]

Switch: [cond\*, stmt\*]

# cond ? iftrue : iffalse

#

TernaryOp: [cond\*, iftrue\*, iffalse\*]

# A base type declaration

#

TypeDecl: [declname, quals, type\*]

# A typedef declaration.

# Very similar to Decl, but without some attributes

#

Typedef: [name, quals, storage, type\*]

Typename: [name, quals, type\*]

UnaryOp: [op, expr\*]

# name: union tag name

# decls: declaration of members

#

Union: [name, decls\*\*]

While: [cond\*, stmt\*]

Pragma: [string]