

C b AST Format

The provided code implements the AST representation described below.

Data Types

The **AST** type is an abstract class which provides a tag and a source code line number. The tag identifies the kind of node; the line number is stored for use in error messages.

There are three subclasses of the **AST** class:

- **AST_nonleaf** used for node types which have a fixed number of children (but at least one).
- **AST_kary** used for node types which have a variable number of children, from zero up.
- **AST_leaf** used for node types which can *never* have any children.

Static factory methods in the AST base class are normally used to instantiate AST nodes. These are as follows.

- `AST.NonLeaf(NodeType tag, int ln, ...)`
- `AST.AST_kary Kary(NodeType tag, int ln, ...)`
- `AST.Leaf(NodeType tag, int ln, string s)`
- `AST.Leaf(NodeType tag, int ln, int i)`
- `AST.Leaf(NodeType tag, int ln)`

In each case, the first two parameters are the tag and the source code line number to be stored in the node. For the **NonLeaf** method, there are one or more additional arguments of type **AST** which are references to the children of this new node. The children (i.e. subtrees) must have already been constructed. The **AST.Kary** method constructs a new node of the **AST_kary** type. There are normally zero children when the node is created ... each subtree is constructed later and added as a child of this node by invoking its **AddChild** method. However, if there are some initial children already constructed, they can be provided as additional arguments to the **Kary** method.

There are three versions of the **AST.Leaf** method. The one with a **string** argument is used when the node has an associated **string** value (such as an identifier or a string constant). The one with an extra **int** argument is used when the node has as associated **int** value (such as an **int** constant). The version with no additional argument is used when the leaf node has neither an associated **int** or **string** value (such as the value **null**).

AST Node Tag Type

The tags are implemented as values of an enum type named **NodeType**. Its constants have the following names.

```
Program, UsingList, DeclList, Const, Struct, Method, FieldList,
FieldDecl, IdList, Formals, Formal, Array, Block, LocalDecl, Assign,
Call, Actuals, PlusPlus, MinusMinus, If, While, Break, Return, Read,
Empty, Add, Sub, Mul, Div, Mod, And, Or, Equals, NotEquals, LessThan,
GreaterThan, LessOrEqual, GreaterOrEqual, UnaryMinus, Dot, NewStruct,
NewArray, Index, IntConst, StringConst, Ident
```

AST Node Types

If the **Arity** entry shows 0, then the **Tag** is used for a leaf node type. If it appears as k , then the **Tag** is used for a k -ary node type.

Tag	Arity	Description
Program	3	Used at root of AST. Children are a using-list, the class name identifier, a list of declarations.
UsingList	k	Children are 0 or more identifiers which were declared after the keyword using .
DeclList	k	Children are 0 or more const, struct or method declarations.
Const	3	A constant declaration; children are the type, the identifier and the value.
Struct	2	A struct declaration; children are the identifier and a list of fields.
FieldList	k	Children are the fields declared inside a struct.
FieldDecl	2	Children are the type and a list of identifiers
IdList	k	Children are 0 or more identifiers
Method	4	A method declaration; children are the typename identifier, the method name identifier, a list of formal parameters, and a method body (a block). If the method has the void type, the first child is missing, i.e. the child reference is null.
Formals	k	Children are 0 or more formal parameter declarations
Formal	2	Children are a type and an identifier (the name of the formal).
Array	1	Represents an array type; the child is the element type.
Block	k	Children are 0 or more local declarations or statements
LocalDecl	2	Children are a type and a list of identifiers
Assign	2	An assignment statement; children are the LHS and the RHS
Call	2	A method call; children are the method name and the arguments (actual parameters).
Actuals	k	Children are 0 or expressions (to be used as actual parameters).
PlusPlus MinusMinus	1	The child is the operand of a postfix ++ or -- operator.
If	3	An if statement; children are the test expression, the then-part statement and the else-part statement.

Tag	Arity	Description
while	2	A while statement; children are the test expression and the statement forming the body.
Break	0	A break statement.
Return	k	A return statement; there is a single optional child, an expression.
Read	2	This is the statement cbio.read(out v); The two children should be the method 'cbio.read' and the variable v .
Empty	0	This is an empty statement. It must also be generated when the <i>else</i> part of an <i>if-statement</i> is omitted.
Add Sub Mul Div Mod	2	An expression where the operator is the binary + - * / or %
And Or	2	An expression where the operator is && or
Equals NotEquals LessThan GreaterThan LessOrEqual GreaterOrEqual	2	An expression where the operator is == != < > <= >=
UnaryMinus	1	An expression where the operator is a prefix -
Dot	2	The LHS is an expression and the RHS is an identifier.
NewStruct	1	The child is an identifier providing the name of a struct type.
NewArray	2	The children are the element type of the array and the size.
Index	2	The children are an expression (which is an array or string value) and an index expression
IntConst StringConst	0	Constants
Ident	0	An identifier (which could represent a type, method, variable or property)