package ast

```
import "github.com/mewmew/uc/ast"
```

Package ast declares the types used to represent abstract syntax trees of µC soure code.

Index

Package Files

ast.go types.go

type ArrayType

```
type ArrayType struct {
    // Element type.
    Elem Type
    // Position of left-bracket `[`.
    Lbracket int
    // Array length.
    Len int
    // Position of right-bracket `]`.
    Rbracket int
}
```

An ArrayType node represents an array type.

Examples.

```
int[]
char[128]
```

func (*ArrayType) Start

```
func (n *ArrayType) Start() int
```

Start returns the start position of the node within the input stream.

func (*ArrayType) String

```
func (n *ArrayType) String() string
```

type BasicLit

```
type BasicLit struct {
    // Position of basic literal.
    ValPos int
    // Basic literal type, one of the following.
    //
    //    token.CharLit
    //    token.IntLit
    Kind token.Kind
    // Basic literal value; e.g. 123, 'a'.
    Val string
}
```

A BasicLit node represents a basic literal.

Examples.

```
42
'a'
```

func (*BasicLit) Start

```
func (n *BasicLit) Start() int
```

Start returns the start position of the node within the input stream.

func (*BasicLit) String

```
func (n *BasicLit) String() string
```

type BinaryExpr

```
type BinaryExpr struct {
    // First operand.
    X   Expr
    // Position of operator.
    OpPos int
    // Operator, one of the following.
    //    token.Add      // +
    //    token.Sub      // -
    //    token.Mul      // *
    //    token.Div      // /
    //    token.Lt       // <
    //    token.Gt       // >
    //    token.Le       // <=
    //    token.Ge       // >=
    //    token.Ne       // !=
    //    token.Eq       // ==
    //    token.Land     // &&
    //    token.Assign   // =
    Op  token.Kind
    // Second operand.
    Y   Expr
}
```

An BinaryExpr node represents a binary expression; X op Y.

Examples.

```
x + y
x = 42
```

func (*BinaryExpr) Start

```
func (n *BinaryExpr) Start() int
```

Start returns the start position of the node within the input stream.

func (*BinaryExpr) String

```
func (n *BinaryExpr) String() string
```

type BlockItem

```
type BlockItem interface {
    Node
    // contains filtered or unexported methods
}
```

A BlockItem represents an item of a block statement, and has one of the following underlying types.

```
Decl
Stmt
```

type BlockStmt

```
type BlockStmt struct {
    // Position of left-brace `{`.
    Lbrace int
    // List of block items contained within the block.
    Items []BlockItem
    // Position of right-brace `}`.
    Rbrace int
}
```

A BlockStmt node represents a block statement.

Examples.

```
{}
{ int x; x = 42; }
```

func (*BlockStmt) Start

```
func (n *BlockStmt) Start() int
```

Start returns the start position of the node within the input stream.

func (*BlockStmt) String

```
func (n *BlockStmt) String() string
```

type CallExpr

```
type CallExpr struct {
    // Function name.
    Name *Ident
    // Position of left-parenthesis `(`.
    Lparen int
    // Function arguments.
    Args []Expr
    // Position of right-parenthesis `)`.
    Rparen int
}
```

A CallExpr node represents a call expression.

Examples.

```
foo()
bar(42)
```

func (*CallExpr) Start

```
func (n *CallExpr) Start() int
```

Start returns the start position of the node within the input stream.

func (*CallExpr) String

```
func (n *CallExpr) String() string
```

type Decl

```
type Decl interface {
    Node
    // Type returns the type of the declared identifier.
    Type() types.Type
    // Name returns the name of the declared identifier.
    Name() *Ident
    // Value returns the initializing value of the defined identifier; or nil if
    // declaration or tentative definition.
    //
    // Underlying type for function declarations.
    //
    //    *BlockStmt
    //
    // Underlying type for variable declarations.
    //
    //    Expr
    //
    // Underlying type for type definitions.
    //
    //    Type
    Value() Node
    // contains filtered or unexported methods
}
```

A Decl node represents a declaration, and has one of the following underlying types.

```
*FuncDecl
*VarDecl
*TypeDef
```

Pseudo-code representation of a declaration.

```
type ident [= value]
```

type EmptyStmt

```
type EmptyStmt struct {
    // Position of semicolon `;`.
    Semicolon int
}
```

An EmptyStmt node represents an empty statement (i.e. ";").

Examples.

```
;
```

func (*EmptyStmt) Start

```
func (n *EmptyStmt) Start() int
```

Start returns the start position of the node within the input stream.

func (*EmptyStmt) String

```
func (n *EmptyStmt) String() string
```

type Expr

```
type Expr interface {
    Node
    // contains filtered or unexported methods
}
```

An Expr node represents an expression, and has one of the following underlying types.

```
*BasicLit
*BinaryExpr
*CallExpr
*Ident
*IndexExpr
*ParenExpr
*UnaryExpr
```

type ExprStmt

```
type ExprStmt struct {
    // Stand-alone expression.
    X Expr
}
```

An ExprStmt node represents a stand-alone expression in a statement list.

Examples.

```
42;
f();
```

func (*ExprStmt) Start

```
func (n *ExprStmt) Start() int
```

Start returns the start position of the node within the input stream.

func (*ExprStmt) String

```
func (n *ExprStmt) String() string
```

type File

```
type File struct {
    // Top-level declarations.
    Decls []Decl
}
```

A File represents a µC source file.

Examples.

```
int x; int main(void) { x = 42; return x; }
```

func (*File) Start

```
func (n *File) Start() int
```

Start returns the start position of the node within the input stream.

func (*File) String

```
func (n *File) String() string
```

type FuncDecl

```
type FuncDecl struct {
    // Function signature.
    FuncType *FuncType
    // Function name.
    FuncName *Ident
    // Function body; or nil if function declaration (i.e. not function
    // definition).
    Body *BlockStmt
}
```

A FuncDecl node represents a function declaration.

Examples.

```
int puts(char s[]);
int add(int a, int b) { return a+b; }
```

func (*FuncDecl) Name

```
func (n *FuncDecl) Name() *Ident
```

Name returns the name of the declared identifier.

func (*FuncDecl) Start

```
func (n *FuncDecl) Start() int
```

Start returns the start position of the node within the input stream.

func (*FuncDecl) String

```
func (n *FuncDecl) String() string
```

func (*FuncDecl) Type

```
func (n *FuncDecl) Type() types.Type
```

Type returns the type of the declared identifier.

func (*FuncDecl) Value

```
func (n *FuncDecl) Value() Node
```

Value returns the initializing value of the defined identifier; or nil if declaration or tentative definition.

Underlying type for function declarations.

```
*BlockStmt
```

type FuncType

```
type FuncType struct {
    // Return type.
    Result Type
    // Position of left-parenthesis `(`.
    Lparen int
    // Function parameters.
    Params []*VarDecl
    // Position of right-parenthesis `)`.
    Rparen int
}
```

A FuncType node represents a function signature.

Examples.

```
int(void)
int(int a, int b)
```

func (*FuncType) Start

```
func (n *FuncType) Start() int
```

Start returns the start position of the node within the input stream.

func (*FuncType) String

```
func (n *FuncType) String() string
```

type Ident

```
type Ident struct {
    // Position of identifier.
    NamePos int
    // Identifier name.
    Name string
    // Corresponding function, variable or type definition. The declaration
    // mapping is added during the semantic analysis phase, based on the
    // lexical scope of the identifier.
    Decl Decl
}
```

An Ident node represents an identifier.

Examples.

```
x
int
```

func (*Ident) Start

```
func (n *Ident) Start() int
```

Start returns the start position of the node within the input stream.

func (*Ident) String

```
func (n *Ident) String() string
```

type IfStmt

```
type IfStmt struct {
    // Position of `if` keyword.
    If  int
    // Condition.
    Cond Expr
    // True branch.
    Body Stmt
    // False branch; or nil if 1-way conditional.
    Else Stmt
}
```

An IfStmt node represents an if statement.

Examples.

```
if (x != 0) { x++; }
if (i < max) { i; } else { max; }
```

func (*IfStmt) Start

```
func (n *IfStmt) Start() int
```

Start returns the start position of the node within the input stream.

func (*IfStmt) String

```
func (n *IfStmt) String() string
```

type IndexExpr

```
type IndexExpr struct {
    // Array name.
    Name *Ident
    // Position of left-bracket `[`.
    Lbracket int
    // Array index.
    Index Expr
    // Position of right-bracket `]`.
    Rbracket int
}
```

An IndexExpr node represents an array index expression.

Examples.

```
buf[i]
```

func (*IndexExpr) Start

```
func (n *IndexExpr) Start() int
```

Start returns the start position of the node within the input stream.

func (*IndexExpr) String

```
func (n *IndexExpr) String() string
```

type Node

```
type Node interface {
    fmt.Stringer
    // Start returns the start position of the node within the input stream.
    Start() int
}
```

A Node represents a node within the abstract syntax tree, and has one of the following underlying types.

```
*File
Decl
Stmt
Expr
Type
```

type ParenExpr

```
type ParenExpr struct {
    // Position of left-parenthesis `(`.
    Lparen int
    // Parenthesised expression.
    X   Expr
    // Position of right-parenthesis `)`.
    Rparen int
}
```

A ParenExpr node represents a parenthesised expression.

func (*ParenExpr) Start

```
func (n *ParenExpr) Start() int
```

Start returns the start position of the node within the input stream.

func (*ParenExpr) String

```
func (n *ParenExpr) String() string
```

type ReturnStmt

```
type ReturnStmt struct {
    // Position of `return` keyword.
    Return int
    // Result expression; or nil if void return.
    Result Expr
}
```

A ReturnStmt node represents a return statement.

Examples.

```
return;
return 42;
```

func (*ReturnStmt) Start

```
func (n *ReturnStmt) Start() int
```

Start returns the start position of the node within the input stream.

func (*ReturnStmt) String

```
func (n *ReturnStmt) String() string
```

type Stmt

```
type Stmt interface {
    Node
    // contains filtered or unexported methods
}
```

A Stmt node represents a statement, and has one of the following underlying types.

```
*BlockStmt
*EmptyStmt
*ExprStmt
*IfStmt
*ReturnStmt
*WhileStmt
```

type Type

```
type Type interface {
    Node
    // contains filtered or unexported methods
}
```

A Type node represents a type of μC, and has one of the following underlying types.

```
*ArrayType
*FuncType
*Ident
```

type TypeDef

```
type TypeDef struct {
    // Position of `typedef` keyword.
    Typedef int
    // Underlying type of type definition.
    DeclType Type
    // Type name.
    TypeName *Ident
    // Underlying type of type definition.
    Val types.Type
}
```

A TypeDef node represents a type definition.

Examples.

```
typedef int foo;
```

func (*TypeDef) Name

```
func (n *TypeDef) Name() *Ident
```

Name returns the name of the declared identifier.

func (*TypeDef) Start

```
func (n *TypeDef) Start() int
```

Start returns the start position of the node within the input stream.

func (*TypeDef) String

```
func (n *TypeDef) String() string
```

func (*TypeDef) Type

```
func (n *TypeDef) Type() types.Type
```

Type returns the type of the declared identifier.

func (*TypeDef) Value

```
func (n *TypeDef) Value() Node
```

Value returns the initializing value of the defined identifier; or nil if declaration or tentative definition.

Underlying type for type definitions.

```
Type
```

type UnaryExpr

```
type UnaryExpr struct {
    // Position of unary operator.
    OpPos int
    // Operator, one of the following.
    //    token.Sub   // -
    //    token.Not   // !
    Op  token.Kind
    // Operand.
    X   Expr
}
```

An UnaryExpr node represents an unary expression; op X.

Examples.

```
-42
!(x == 3 || x == 10)
```

func (*UnaryExpr) Start

```
func (n *UnaryExpr) Start() int
```

Start returns the start position of the node within the input stream.

func (*UnaryExpr) String

```
func (n *UnaryExpr) String() string
```

type VarDecl

```
type VarDecl struct {
    // Variable type.
    VarType Type
    // Variable name.
    VarName *Ident
    // Variable value expression; or nil if variable declaration (i.e. not
    // variable definition).
    Val Expr
}
```

A VarDecl node represents a variable declaration.

Examples.

```
int x;
char buf[128];
```

func (*VarDecl) Name

```
func (n *VarDecl) Name() *Ident
```

Name returns the name of the declared identifier.

func (*VarDecl) Start

```
func (n *VarDecl) Start() int
```

Start returns the start position of the node within the input stream.

func (*VarDecl) String

```
func (n *VarDecl) String() string
```

func (*VarDecl) Type

```
func (n *VarDecl) Type() types.Type
```

Type returns the type of the declared identifier.

func (*VarDecl) Value

```
func (n *VarDecl) Value() Node
```

Value returns the initializing value of the defined identifier; or nil if declaration or tentative definition.

Underlying type for variable declarations.

```
Expr
```

type WhileStmt

```
type WhileStmt struct {
    // Position of `while` keyword.
    While int
    // Condition.
    Cond Expr
    // Loop body.
    Body Stmt
}
```

A WhileStmt node represents a while statement.

Examples.

```
while (i < 10) { i++; }
```

func (*WhileStmt) Start

```
func (n *WhileStmt) Start() int
```

Start returns the start position of the node within the input stream.

func (*WhileStmt) String

```
func (n *WhileStmt) String() string
```

Directories

Path	Synopsis
astutil	Package astutil implements utility functions for handling parse trees.
astx	Package astx implements utility functions for generating abstract syntax trees.