Academic C

C subset implementation

What it does

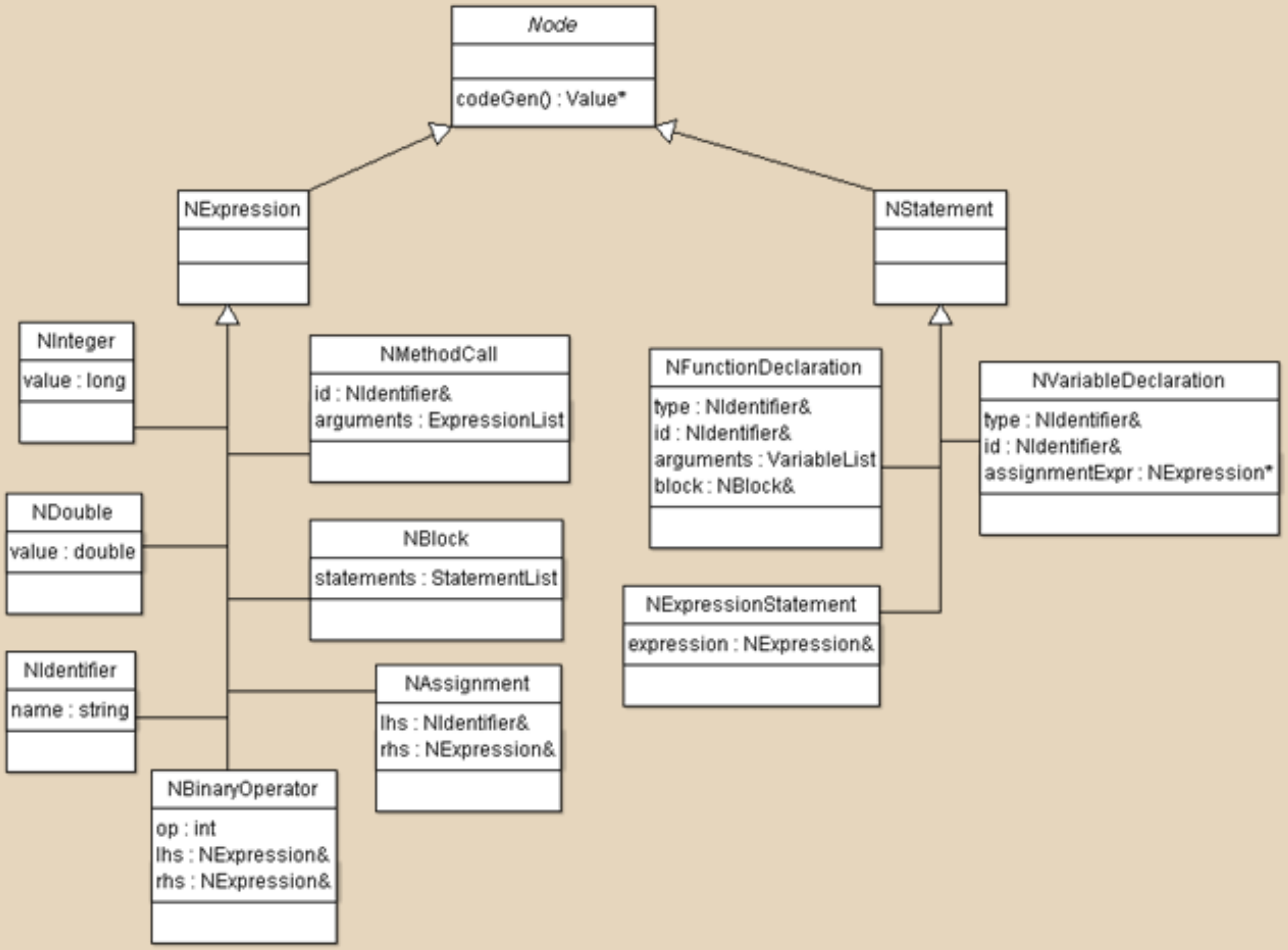
- **variables**
- **types**: int / double
- **expressions:** arithmetic / logic
- **functions:** definition / call
- **branching** (if / else)
- **printf**(fmt, ...)

How it's made

flex	lexer
byson	parser
llvm 2.9	code generation / assember
gcc	asm compiler
make	auto build
love	

Architecture Overview

STEP	RESULT
Source Code	Stream of text
Lexer	Stream of tokens
Parser	Abstract Syntax Tree
Codegen	LLVM Syntax Tree / LLVM asm
llvm-as	LLVM bitcode
llc	native assembly
gcc	native binary



Node
codeGen() : Value*
NExpression
NStatement
NInteger
value : long
NMethodCall
id : NIdentifier&
arguments : ExpressionList
NDouble
value : double
NBlock
statements : StatementList
NIdentifier
name : string
NAssignment
lhs : NIdentifier&
rhs : NExpression&
NBinaryOperator
op : int
lhs : NExpression&
rhs : NExpression&
NFunctionDeclaration
type : NIdentifier&
id : NIdentifier&
arguments : VariableList
block : NBlock&
NVariableDeclaration
type : NIdentifier&
id : NIdentifier&
assignmentExpr : NExpression*
NExpressionStatement
expression : NExpression&

How to build it

make **lft-cc**-> lft-cc

Builds the lexer, parser and codegen.

make **run** -> out.ll

Runs the compiler on the dummy source code.

make **llvm-as** -> out.bc (use xxd)

Runs the llvm assembler on the out.ll file.

make **llc** -> out.s (native assembly file)

Runs the LLVM compiler on the out.bc file.

make **native-compiler** -> out

Runs the gcc compiler on the out.s assembly

Example

CODE:

```
int fibo( int n ){
    if( n < 3 ){
        return 1;
    } else {
        return fibo( n - 1 ) + fibo( n - 2 );
    };
};
double first_fibo( int count ){
    if( count > 0 ){
        first_fibo( count - 1 );
        printf( "Fibo[%d] = %d\n", count, fibo(count) );
        return 0.0;
    } else { return 0.0; };
};
first_fibo( 9 );
```

OUTPUT:

```
Fibo[1] = 1
Fibo[2] = 1
Fibo[3] = 2
Fibo[4] = 3
Fibo[5] = 5
Fibo[6] = 8
Fibo[7] = 13
Fibo[8] = 21
Fibo[9] = 34
```

What now?

Seriously, have you seen the standard?

- full type support (arrays, pointers, chars ..)
- error checking,
- structs, unions,
- fixed branching,
- multiple sources,
- full operator support,
- loops,
- name resolution

...

Bibliography

Writing Your Own Toy Compiler Using Flex, Bison and LLVM
http://gnuu.org/2009/09/18/writing-your-own-toy-compiler/4/

LLVM Documentation
http://llvm.org/docs/doxygen/html/

Kaleidoscope: Implementing a Language with LLVM
http://llvm.org/docs/tutorial/index.html