

Derivation of LL(1) Grammar for C0

Stochastic Batman

January 1st, 2026

1 Original Context-Free Grammar of C0

The original grammar is presented below, with dereference operator changed from * to @.

1.1 Lexical Rules

Non-terminal	Production	Description
<Di>	0 1 2 3 4 5 6 7 8 9	Digit
<DiS>	<Di> <Di><DiS>	Digit sequence
<Le>	a ... z A ... Z _	Letter
<DiLe>	<Le> <Di>	Alphanumeric symbol
<DiLeS>	<DiLe> <DiLe><DiLeS>	Sequence of alphanumeric symbols
<Na>	<Le> <Le><DiLeS>	Name
<C>	<DiS> <DiS>u null	int/uint/null constant
<CC>	'_-' ... '^'	Char-constant with ASCII code
<BC>	true false	Bool-constant
<id>	<Na> <id>. <Na> <id>[<E>] <id>@ <id>&	Identifier (field, index, deref, addr)

1.2 Expressions

Non-terminal	Production	Description
<F>	<id> -<F> (<E>) <C>	Factor
<T>	<F> <T>*<F> <T>/<F>	Term
<E>	<T> <E>+<T> <E>-<T>	Expression
<Atom>	<E> > <E> <E> >= <E> <E> < <E> <E> <= <E> <E> == <E> <E> != <E> <BC>	Atom
<BF>	<id> <Atom> !<BF> (<BE>)	Boolean factor
<BT>	<BF> <BT> && <BF>	Boolean term
<BE>	<BT> <BE> <BT>	Boolean expression
<Pa>	<E> <BE> <CC>	Parameter

$\langle \text{PaS} \rangle$	$\langle \text{Pa} \rangle \mid \langle \text{Pa} \rangle, \langle \text{PaS} \rangle$	Parameter sequence
------------------------------	--	--------------------

1.3 Statements

Non-terminal	Production	Description
$\langle \text{St} \rangle$	$\langle \text{id} \rangle = \langle \text{E} \rangle \mid \langle \text{id} \rangle = \langle \text{BE} \rangle \mid \langle \text{id} \rangle = \langle \text{CC} \rangle \mid \text{if } \langle \text{BE} \rangle \{ \langle \text{StS} \rangle \} \mid \text{if } \langle \text{BE} \rangle \{ \langle \text{StS} \rangle \} \text{ else } \{ \langle \text{StS} \rangle \} \mid \text{while } \langle \text{BE} \rangle \{ \langle \text{StS} \rangle \} \mid \langle \text{id} \rangle = \langle \text{Na} \rangle (\langle \text{PaS} \rangle) \mid \langle \text{id} \rangle = \langle \text{Na} \rangle () \mid \langle \text{id} \rangle = \text{new } \langle \text{Na} \rangle @$	Assignment / if-then / if-then-else / while / call / alloc
$\langle \text{StS} \rangle$	$\langle \text{St} \rangle \mid \langle \text{St} \rangle; \langle \text{StS} \rangle$	Statement sequence
$\langle \text{rSt} \rangle$	$\text{return } \langle \text{E} \rangle \mid \text{return } \langle \text{BE} \rangle \mid \text{return } \langle \text{CC} \rangle$	Return statement

1.4 Types and Declarations

Non-terminal	Production	Description
$\langle \text{Ty} \rangle$	$\text{int} \mid \text{bool} \mid \text{char} \mid \text{uint} \mid \langle \text{Na} \rangle$	Basic type
$\langle \text{VaD} \rangle$	$\langle \text{Ty} \rangle \langle \text{Na} \rangle$	Variable declaration
$\langle \text{VaDS} \rangle$	$\langle \text{VaD} \rangle \mid \langle \text{VaD} \rangle; \langle \text{VaDS} \rangle$	Variable declaration sequence
$\langle \text{TE} \rangle$	$\langle \text{Ty} \rangle [\langle \text{DiS} \rangle] \mid \langle \text{Ty} \rangle @ \mid \text{struct } \{ \langle \text{VaDS} \rangle \}$	Type expression (array/pointer/struct)
$\langle \text{TyD} \rangle$	$\text{typedef } \langle \text{TE} \rangle \langle \text{Na} \rangle$	Type declaration
$\langle \text{TyDS} \rangle$	$\langle \text{TyD} \rangle \mid \langle \text{TyD} \rangle; \langle \text{TyDS} \rangle$	Type declaration sequence

1.5 Functions and Program

Non-terminal	Production	Description
$\langle \text{body} \rangle$	$\langle \text{rSt} \rangle \mid \langle \text{StS} \rangle; \langle \text{rSt} \rangle$	Function body
$\langle \text{PaDS} \rangle$	$\langle \text{VaD} \rangle \mid \langle \text{VaD} \rangle, \langle \text{PaDS} \rangle$	Parameter declarations
$\langle \text{FuD} \rangle$	$\langle \text{Ty} \rangle \langle \text{Na} \rangle (\langle \text{PaDS} \rangle) \{ \langle \text{VaDS} \rangle; \langle \text{body} \rangle \} \mid \langle \text{Ty} \rangle \langle \text{Na} \rangle (\langle \text{PaDS} \rangle) \{ \langle \text{body} \rangle \} \mid \langle \text{Ty} \rangle \langle \text{Na} \rangle () \{ \langle \text{VaDS} \rangle; \langle \text{body} \rangle \} \mid \langle \text{Ty} \rangle \langle \text{Na} \rangle () \{ \langle \text{body} \rangle \}$	Function declaration
$\langle \text{FuDS} \rangle$	$\langle \text{FuD} \rangle \mid \langle \text{FuD} \rangle; \langle \text{FuDS} \rangle$	Function sequence
$\langle \text{prog} \rangle$	$\langle \text{TyDS} \rangle; \langle \text{VaDS} \rangle; \langle \text{FuDS} \rangle \mid \langle \text{VaDS} \rangle; \langle \text{FuDS} \rangle \mid \langle \text{TyDS} \rangle; \langle \text{FuDS} \rangle \mid \langle \text{FuDS} \rangle$	Program

2 Deriving the LL(1) Grammar

To transform the original grammar into an LL(1) grammar, I address issues such as left recursion, common prefixes in alternatives, and ambiguity. I proceed section by section, deriving each non-terminal in the LL(1) version from the original rules. I introduce new non-terminals for tails to handle recursion and factoring.

2.1 Lexical Rules and Terminals

The lexical rules are mostly regular expressions and not recursive, so they are largely unchanged. In the LL(1) version, I use terminals like ID for $\langle Na \rangle$, NUM for $\langle DiS \rangle$, etc.

I refactor the identifier $\langle id \rangle$ into l-value with postfix operators.

Original $\langle id \rangle : \langle Na \rangle \mid \langle id \rangle . \langle Na \rangle \mid \langle id \rangle [\langle E \rangle] \mid \langle id \rangle @ \mid \langle id \rangle &$.

This has left recursion. To eliminate it, I introduce $\langle lvalue \rangle$ and $\langle lvalue_tail \rangle$.

Derive $\langle lvalue \rangle$:

I start with base: $\langle lvalue \rangle \rightarrow ID \langle lvalue_tail \rangle$, where ID replaces $\langle Na \rangle$.

Now, the recursive parts: $\cdot \mid ID \mid [\langle Expr \rangle] \mid @ \mid &$, and ϵ for no more.

So, $\langle lvalue_tail \rangle \rightarrow \cdot \mid ID \langle lvalue_tail \rangle \mid [\langle Expr \rangle] \langle lvalue_tail \rangle \mid @ \langle lvalue_tail \rangle \mid & \langle lvalue_tail \rangle \mid \epsilon$.

This is left-factored already since each alternative starts with a distinct terminal: $\cdot, [, @, &, \text{ or } \epsilon$. No left recursion because tail is after the operator. This matches the LL(1) version.

For constants, $\langle C \rangle, \langle CC \rangle, \langle BC \rangle$ become terminals.

2.2 Types and Declarations

I start with types.

Original $\langle Ty \rangle : \text{int} \mid \text{bool} \mid \text{char} \mid \text{uint} \mid \langle Na \rangle$.

I replace $\langle Na \rangle$ with ID, so $\langle Ty \rangle \rightarrow \text{int} \mid \text{bool} \mid \text{char} \mid \text{uint} \mid ID$. Unchanged.

Original $\langle TE \rangle : \langle Ty \rangle [\langle DiS \rangle] \mid \langle Ty \rangle @ \mid \text{struct} \{ \langle VaDS \rangle \}$.

I replace $\langle DiS \rangle$ with NUM. This has common prefix $\langle Ty \rangle$ in first two alternatives.

Left-factor: I introduce $\langle TEprime \rangle$ for the modifier.

$\langle TE \rangle \rightarrow \langle Ty \rangle \langle TEprime \rangle \mid \text{struct} \{ \langle VaDS \rangle \}$.

$\langle TEprime \rangle \rightarrow [\text{NUM}] \mid @ \mid \epsilon$.

Now distinct starts: int/bool/char/uint/ID vs struct. Within $\langle TEprime \rangle$, starts with $[, @, \text{ or } \epsilon$.
Matches LL(1).

Variable declarations:

Original $\langle VaD \rangle : \langle Ty \rangle \langle Na \rangle \rightarrow \langle Ty \rangle ID$. Unchanged.

Original $\langle VaDS \rangle : \langle VaD \rangle \mid \langle VaD \rangle ; \langle VaDS \rangle$. Left recursive.

Eliminate: $\langle VaDS \rangle \rightarrow \langle VaD \rangle \langle VaDS_tail \rangle$.

$\langle VaDS_tail \rangle \rightarrow ; \mid \langle VaD \rangle \langle VaDS_tail \rangle \mid \epsilon$.

Starts with ; or ϵ . Matches LL(1).

Type declarations follow similarly:

$\langle TyDS \rangle \rightarrow \langle TyD \rangle \langle TyDS_tail \rangle$.

$\langle TyDS_tail \rangle \rightarrow ; \mid \langle TyD \rangle \langle TyDS_tail \rangle \mid \epsilon$.

Optional typedefs: $\langle TDS0 \rangle \rightarrow \langle TyDS \rangle \mid \epsilon$.

2.3 Expressions

The expressions are separated into arithmetic and boolean in the original grammar, but in LL(1), I combine them into $\langle Expr \rangle$ with precedence levels.

Multiplicative expressions:

Original $\langle T \rangle \rightarrow \langle F \rangle \mid \langle T \rangle * \langle F \rangle \mid \langle T \rangle / \langle F \rangle$. This is left recursive.

I remove it: $\langle MulExpr \rangle \rightarrow \langle Primary \rangle \langle MulExpr_tail \rangle$.

$\langle MulExpr_tail \rangle \rightarrow * \mid \langle Primary \rangle \langle MulExpr_tail \rangle \mid / \mid \langle Primary \rangle \langle MulExpr_tail \rangle \mid \epsilon$.

Additive expressions:

Original $\langle E \rangle \rightarrow \langle T \rangle \mid \langle E \rangle + \langle T \rangle \mid \langle E \rangle - \langle T \rangle$.

I transform it to: $\langle AddExpr \rangle \rightarrow \langle MulExpr \rangle \langle AddExpr_tail \rangle$.

$\langle AddExpr_tail \rangle \rightarrow + \mid \langle MulExpr \rangle \langle AddExpr_tail \rangle \mid - \mid \langle MulExpr \rangle \langle AddExpr_tail \rangle \mid \epsilon$.

Relational expressions:

I create: $\langle RelExpr \rangle \rightarrow \langle AddExpr \rangle \langle RelExpr_tail \rangle$.

$\langle RelExpr_tail \rangle \rightarrow \langle rel_op \rangle \mid \langle AddExpr \rangle \langle RelExpr_tail \rangle \mid \epsilon$.

Boolean expressions:

I build the AND level: $\langle \text{AndExpr} \rangle \rightarrow \langle \text{RelExpr} \rangle \langle \text{AndExpr_tail} \rangle$.
 $\langle \text{AndExpr_tail} \rangle \rightarrow \&& \langle \text{RelExpr} \rangle \langle \text{AndExpr_tail} \rangle \mid \epsilon$.
And the OR level: $\langle \text{Expr} \rangle \rightarrow \langle \text{AndExpr} \rangle \langle \text{Expr_tail} \rangle$.
 $\langle \text{Expr_tail} \rangle \rightarrow \mid\mid \langle \text{AndExpr} \rangle \langle \text{Expr_tail} \rangle \mid \epsilon$.

2.4 Statements

Original $\langle \text{St} \rangle$ has many alternatives. I unify them into:

$\langle \text{lvalue} \rangle = \langle \text{RHS} \rangle \mid \text{if } \langle \text{Expr} \rangle \{ \langle \text{StS} \rangle \} \langle \text{EP} \rangle \mid \text{while } \langle \text{Expr} \rangle \{ \langle \text{StS} \rangle \}$.
Where I define $\langle \text{RHS} \rangle \rightarrow \langle \text{Expr} \rangle \mid \text{new ID } @$.

For if-then-else, I left factor by introducing:

$\langle \text{EP} \rangle \rightarrow \text{else } \{ \langle \text{StS} \rangle \} \mid \epsilon$.

Statement sequences:

Original $\langle \text{StS} \rangle: \langle \text{St} \rangle \mid \langle \text{St} \rangle ; \langle \text{StS} \rangle$. Left recursive.

I remove it: $\langle \text{StS} \rangle \rightarrow \langle \text{St} \rangle \langle \text{StS_tail} \rangle$.

$\langle \text{StS_tail} \rangle \rightarrow ; \langle \text{St} \rangle \langle \text{StS_tail} \rangle \mid \epsilon$.

For return: I use `return` $\langle \text{Expr} \rangle$, since I've unified the expression types.

2.5 Function Body

Original body: $\langle \text{rSt} \rangle \mid \langle \text{StS} \rangle ; \langle \text{rSt} \rangle$.

In LL(1), I define: $\langle \text{body} \rangle \rightarrow \langle \text{SS0} \rangle \langle \text{rSt} \rangle$.

$\langle \text{SS0} \rangle \rightarrow \langle \text{StS} \rangle \mid \epsilon$.

For locals, I introduce the `local` keyword to distinguish local variable declarations:

$\langle \text{locals} \rangle \rightarrow \text{local } \langle \text{VaDS} \rangle \mid \epsilon$.

2.6 Function Parameters

Original $\langle \text{PaDS} \rangle \rightarrow \langle \text{VaD} \rangle \mid \langle \text{VaD} \rangle , \langle \text{PaDS} \rangle$. Left recursive.

I remove it: $\langle \text{PaDS} \rangle \rightarrow \langle \text{VaD} \rangle \langle \text{PaDS_tail} \rangle$.

$\langle \text{PaDS_tail} \rangle \rightarrow , \langle \text{VaD} \rangle \langle \text{PaDS_tail} \rangle \mid \epsilon$.

I define $\langle \text{PDS0} \rangle \rightarrow \langle \text{PaDS} \rangle \mid \epsilon$ for empty () .

2.7 Program and Globals

In LL(1), I unify the program structure as optional $\langle \text{TDS0} \rangle$ followed by $\langle \text{GDs} \rangle$.

I define: $\langle \text{GD} \rangle \rightarrow \langle \text{Ty} \rangle \text{ ID } \langle \text{GDT} \rangle$.

$\langle \text{GDT} \rangle \rightarrow ; \mid (\langle \text{PDS0} \rangle) \{ \langle \text{locals} \rangle \langle \text{body} \rangle \}$.

This distinguishes global variables ($\langle \text{Ty} \rangle \text{ ID } ;$) from functions ($\langle \text{Ty} \rangle \text{ ID } (\dots) \{ \dots \}$).

I define: $\langle \text{GDs} \rangle \rightarrow \langle \text{GD} \rangle \langle \text{GDs} \rangle \mid \epsilon$.

$\langle \text{prog} \rangle \rightarrow \langle \text{TDS0} \rangle \langle \text{GDs} \rangle$.

3 Derived LL(1) Grammar of C0

The derived LL(1) grammar is as follows.

3.1 Types

Non-terminal	Production	Description
$\langle \text{Ty} \rangle$	int bool char uint ID	Basic type
$\langle \text{TEprime} \rangle$	[NUM] @ ϵ	Type modifier

$\langle TE \rangle$	$\langle Ty \rangle \langle TE' \rangle \mid \text{struct } \{ \langle VaDS \rangle \}$	Type expression
----------------------	---	-----------------

3.2 Variable Declarations

Non-terminal	Production	Description
$\langle VaD \rangle$	$\langle Ty \rangle \text{ ID}$	Variable declaration
$\langle VaDS_tail \rangle$	$; \langle VaD \rangle \langle VaDS_tail \rangle \mid \epsilon$	More var decls
$\langle VaDS \rangle$	$\langle VaD \rangle \langle VaDS_tail \rangle$	Var decl sequence

3.3 Type Declarations

Non-terminal	Production	Description
$\langle TyD \rangle$	$\text{typedef } \langle TE \rangle \text{ ID}$	Type declaration
$\langle TyDS_tail \rangle$	$; \langle TyD \rangle \langle TyDS_tail \rangle \mid \epsilon$	More type decls
$\langle TyDS \rangle$	$\langle TyD \rangle \langle TyDS_tail \rangle$	Type decl sequence
$\langle TDS0 \rangle$	$\langle TyDS \rangle \mid \epsilon$	Optional typedefs

3.4 L-values

Non-terminal	Production	Description
$\langle lvalue \rangle$	$\text{ID } \langle lvalue_tail \rangle$	L-value
$\langle lvalue_tail \rangle$	$. \text{ ID } \langle lvalue_tail \rangle \mid [\langle Expr \rangle] \langle lvalue_tail \rangle \mid @ \langle lvalue_tail \rangle \mid & \langle lvalue_tail \rangle \mid \epsilon$	L-value postfix

3.5 Expressions

Non-terminal	Production	Description
$\langle Primary \rangle$	$\text{ID } \langle primary_tail \rangle \mid - \langle Primary \rangle \mid ! \langle Primary \rangle \mid (\langle Expr \rangle) \mid \langle C \rangle \mid \langle CC \rangle \mid \langle BC \rangle$	Primary expression
$\langle primary_tail \rangle$	$(\langle PSO \rangle) \mid \langle lvalue_tail \rangle$	Call or postfix ops
$\langle MulExpr \rangle$	$\langle Primary \rangle \langle MulExpr_tail \rangle$	Multiplicative expr
$\langle MulExpr_tail \rangle$	$* \langle Primary \rangle \langle MulExpr_tail \rangle \mid / \langle Primary \rangle \langle MulExpr_tail \rangle \mid \epsilon$	Mul/div continuation
$\langle AddExpr \rangle$	$\langle MulExpr \rangle \langle AddExpr_tail \rangle$	Additive expr
$\langle AddExpr_tail \rangle$	$+ \langle MulExpr \rangle \langle AddExpr_tail \rangle \mid - \langle MulExpr \rangle \langle AddExpr_tail \rangle \mid \epsilon$	Add/sub continuation
$\langle RelExpr \rangle$	$\langle AddExpr \rangle \langle RelExpr_tail \rangle$	Relational expr
$\langle RelExpr_tail \rangle$	$\langle rel_op \rangle \langle AddExpr \rangle \langle RelExpr_tail \rangle \mid \epsilon$	Relational continuation
$\langle AndExpr \rangle$	$\langle RelExpr \rangle \langle AndExpr_tail \rangle$	Logical AND expr

$\langle \text{AndExpr_tail} \rangle$	$\&\& \langle \text{RelExpr} \rangle \langle \text{AndExpr_tail} \rangle \mid \epsilon$	AND continuation
$\langle \text{Expr} \rangle$	$\langle \text{AndExpr} \rangle \langle \text{Expr_tail} \rangle$	Full expression
$\langle \text{Expr_tail} \rangle$	$\mid\mid \langle \text{AndExpr} \rangle \langle \text{Expr_tail} \rangle \mid \epsilon$	OR continuation

3.6 Call Parameters

Non-terminal	Production	Description
$\langle \text{PaS} \rangle$	$\langle \text{Expr} \rangle \langle \text{PaS_tail} \rangle$	Parameter sequence
$\langle \text{PaS_tail} \rangle$	$, \langle \text{Expr} \rangle \langle \text{PaS_tail} \rangle \mid \epsilon$	More parameters
$\langle \text{PSO} \rangle$	$\langle \text{PaS} \rangle \mid \epsilon$	Optional parameters

3.7 Statements

Non-terminal	Production	Description
$\langle \text{RHS} \rangle$	$\langle \text{Expr} \rangle \mid \text{new ID } @$	Assignment RHS
$\langle \text{rSt} \rangle$	$\text{return } \langle \text{Expr} \rangle$	Return statement
$\langle \text{EP} \rangle$	$\text{else } \{ \langle \text{StS} \rangle \} \mid \epsilon$	Optional else
$\langle \text{St} \rangle$	$\langle \text{lvalue} \rangle = \langle \text{RHS} \rangle \mid \text{if } \langle \text{Expr} \rangle \{ \langle \text{StS} \rangle \}$	Statement
$\langle \text{StS_tail} \rangle$	$\langle \text{EP} \rangle \mid \text{while } \langle \text{Expr} \rangle \{ \langle \text{StS} \rangle \}$	
$\langle \text{StS_tail} \rangle$	$; \langle \text{St} \rangle \langle \text{StS_tail} \rangle \mid \epsilon$	More statements
$\langle \text{StS} \rangle$	$\langle \text{St} \rangle \langle \text{StS_tail} \rangle$	Statement sequence

3.8 Function Body

Non-terminal	Production	Description
$\langle \text{locals} \rangle$	$\text{local } \langle \text{VaDS} \rangle \mid \epsilon$	Local declarations
$\langle \text{SSO} \rangle$	$\langle \text{StS} \rangle \mid \epsilon$	Optional statements
$\langle \text{body} \rangle$	$\langle \text{SSO} \rangle \langle \text{rSt} \rangle$	Function body

3.9 Function Parameters

Non-terminal	Production	Description
$\langle \text{PaDS} \rangle$	$\langle \text{VaD} \rangle \langle \text{PaDS_tail} \rangle$	Param declarations
$\langle \text{PaDS_tail} \rangle$	$, \langle \text{VaD} \rangle \langle \text{PaDS_tail} \rangle \mid \epsilon$	More param decls
$\langle \text{PDSO} \rangle$	$\langle \text{PaDS} \rangle \mid \epsilon$	Optional param decls

3.10 Program

Non-terminal	Production	Description
$\langle \text{GDT} \rangle$	$; \mid (\langle \text{PDSO} \rangle) \{ \langle \text{locals} \rangle \langle \text{body} \rangle \}$	Var end or function def

$\langle \text{GD} \rangle$	$\langle \text{Ty} \rangle \text{ ID } \langle \text{GDT} \rangle$	Global declaration
$\langle \text{GDs} \rangle$	$\langle \text{GD} \rangle \langle \text{GDs} \rangle \mid \epsilon$	Global decl sequence
$\langle \text{prog} \rangle$	$\langle \text{TDS0} \rangle \langle \text{GDs} \rangle$	Program

4 Terminals

Terminal	Description
ID	Identifier
NUM	Integer literal (for array sizes)
<C>	Integer constant
<CC>	Character constant
<BC>	Boolean constant (<code>true</code> , <code>false</code>)
$\langle \text{rel_op} \rangle$	Relational operator ($<$, $>$, \leq , \geq , $=$, \neq)
int, bool, char, uint	Built-in type keywords
struct, typedef, new	Type-related keywords
if, else, while, return, local	Control and declaration keywords
$+, -, *, /$	Arithmetic operators
$\&&, , !$	Logical operators
$\text{@}, \&$	Pointer dereference and address-of
.	Field access
[,]	Array indexing
(,)	Parentheses
{, }	Braces
;	Statement/declaration separator
,	Parameter separator
=	Assignment
