# The C- Programming Language Grammar

Fall 2022

## 1. Introduction

This document contains a grammar for the C- programming language that will be used for the Fall 2021 semester. This language is similar to C and shares many features in common with real-world programming languages. This grammar contains the classic "dangling else" ambiguity that you will be required to remove.

This grammar is presented in standard BNF notation. Non-terminals are surrounded by < >. Token classes are shown in all caps (e.g. NUMCONST). C- is case sensitive. Whitespace (sequences of blanks and tabs) is ignored. Comments in C- begin with // and continue to the end of the line and are also ignored.

## 1.1 Token Classes

### 1.1.1 The ID Token Class

An identifier (or ID) in C- can be composed of uppercase and lowercase letters (a | .. | z | A | … | Z) and digits (0..9). An identifier must start with a letter.

### 1.1.2 The NUMCONST Token Class

A numeric constant in C- can be composed of 1 or more digits (0..9).

### 1.1.3 The CHARCONST Token Class

A character constant in C- is a representation of a single character that is placed in quotes. For example, the representation for the character constant Z in C- would be 'Z'. C- permits "escape sequences" commonly found in programming languages. In C- there are two valid escape sequences: '\n' and '\0'. These escape sequences have the same interpretation in C- that they have in C. In C- any character can be preceded by a backslash, but unless the character that follows the backslash is n or 0, the escape character does not add any meaning. For example, the character constants 'Z' and '\Z' have the same meaning, whereas the character constants '0' and '\0' do not. In C- the sequence '\\' represents a single backslash character, and the sequence '\'' represents a quote character.

### 1.1.4 The STRINGCONST Token Class

A string constant in C- is any fixed-length sequence of characters enclosed in doublequotes "like this". String constants in C- can contain escape sequences. Since a string is a sequence of characters in C-, escape sequences in string constants are treated the same as escape sequences in character constants. The string constant "" is a valid C- string of length 0. String constants in C- are terminated by the first unescaped doublequote character. The string constant containing a single doublequote character in C- is "\"" (this constant would have a length of 1 in C-). C- string constants must be entirely contained on a single line.

## 2. The Grammar

<program> ::= <declList>

<declList> ::= <declList> <decl> | <decl>

<decl> ::= <varDecl> | <funDecl>

<varDecl> ::= <typeSpec> <varDeclList> **;**

<scopedVarDecl> ::= static <typeSpec> <varDeclList> **;** | <typeSpec> <varDeclList> **;**

<varDeclList> ::= <varDeclList> **,** <varDeclInit> | varDeclInit

<varDeclInit> ::= <varDeclId> | <varDeclId> **:** <simpleExp>

<varDeclId> ::= ID | ID [ NUMCONST ]

<typeSpec> ::= bool | char | int

<funDecl> ::=  <typeSpec> ID ( parms ) compoundStmt | ID ( parms ) compoundStmt

<parms> ::= <parmList> | ε

<parmList> ::= <parmList> **;** <parmTypeList> | parmTypeList

<parmTypeList> ::= <typeSpec> <parmIdList>

<parmIdList> ::= <parmIdList> **,** <parmId> | <parmId>

<parmId> ::=  ID | ID [ ]

<stmt> ::= <expStmt> | <compoundStmt> | <selectStmt> | <iterStmt> | <returnStmt> | <breakStmt>

<expStmt> ::= <exp> ; | ;

<compoundStmt> ::= { <localDecls> <stmtList> }

<localDecls> ::= <localDecls> <scopedVarDecl> | ε

<stmtList> ::= <stmtList> <stmt> |  ε

<selectStmt> ::= if <simpleExp> then <stmt> |  if <simpleExp> then <stmt> else <stmt>

<iterStmt> ::= while <simpleExp> do <stmt> | for ID = <iterRange> do <stmt>

<iterRange> ::= <simpleExp> to <simpleExp> | <simpleExp> to <simpleExp> by <simpleExp>

<returnStmt> ::= return ; | return <exp> ;

<breakStmt> ::= break ;

<exp> ::= <mutable> <assignop> <exp> | <mutable>++ | <mutable>-- | <simpleExp>

<assignop> ::=  = | += | -= | *= | /=

<simpleExp> ::= <simpleExp> or <andExp> | <andExp>

<andExp> ::= <andExp> and <unaryRelExp> | <unaryRelExp>

<unaryRelExp> ::= not <unaryRelExp> | <relExp>

<relExp> ::= <sumExp> <relop> <sumExp> | <sumExp>

<relop> ::= < | <= | > | >= | == | !=

<sumExp> ::= <sumExp> <sumop> <mulExp> | <mulExp>

<sumop> ::= + | -

<mulExp> ::= <mulExp> <mulop> <unaryExp> | <unaryExp>

<mulop> ::= * | / | %

<unaryExp> ::= <unaryop> <unaryExp> | <factor>

<unaryop> ::= - | * | ?

<factor> ::= <mutable> | <immutable>

<mutable> ::= ID  | ID [ <exp> ]

<immutable> ::= ( <exp> )  | <call> | <constant>

<call> ::= ID ( <args> )

<args> ::= <argList> | ε

<argList> ::= <argList> , <exp> | <exp>

<constant> ::= NUMCONST  | CHARCONST  | STRINGCONST | true | false

## 3. Semantics

- The only numbers in C- are integral numbers (**int**s).
- There is no type conversion or coercion in C-. For instance, an entity of type **bool** cannot be converted to or compared with an entity of type **int**.
- Function names must be unique in C- programs. There is no overloading of function names. There is a single namespace for both functions and variables, so functions and variables cannot have the same name in the same scope.
- In C- all variables and functions must be declared before they are used.
- The unary asterisk operator * takes an array as its argument and produces the size of the array.
- The STRINGCONST token represents a fixed-length array of type **char**.
- Logical operators in C- are not short-circuiting.
- In C- **if** statements, the **else** clause is matched with the closest unmatched **if**. This is a relatively standard interpretation used in many programming languages. The grammar in this document contains the classic “dangling else” ambiguity that you will be required to fix.
- The associativity and precedence of C- arithmetic operators is the same as what one would expect in mathematics. No re-ordering of operands when evaluating expressions is permitted.
- In C-, an entity of type **char** occupies the same amount of space as an entity of type **int** or type **bool**. This is an artifact of the machine that we will be targeting.
- Variable initializers in C- must be compile-time constants. Variables cannot be initialized with values that cannot be determined at compile-time. The type of a variable and its initializing expression must match.
- Assignment of arrays is permitted in C-. In array assignments, the contents of the source array is copied into the destination array. If the destination array is smaller than the source array, only as many characters as will fit into the destination are copied. If the source array is smaller than the destination array, all of the elements in the source are copied into the destination, and any remaining elements in the destination that were not overwritten are untouched. The target machine contains support for this operation.
- Assignment of a string to a char array works as though it were any other assignment.
- Arrays are passed into functions by reference in C-. This is implemented via pointers. Functions in C- cannot return an array, but can modify the contents of an array that is passed in as an argument.
- C- permits comparison of arrays.

- Assignment in C- expressions occurs at the time that the assignment operator is encountered when evaluating an expression in its proper order. The value of an assignment expression is the righthand side of the expression. The ++ and - - operators are assignment operators in C- (this is different than in C/C++). Note that assignment in a C- <simpleExp> must be enclosed in parentheses.

- The return type of a C- function is specified in its declaration. If no type is ascribed to a function in its declaration, it is assumed that the function does not return a value.

- Code that exits a function without a specific **return** statement returns a 0 for a function of type int, a false for a function of type **bool**, and the space character (‘ ‘) for a function of type **char**.

- The unary ? operator generates a pseudo-random integer in the range [0..|n|-1] with the sign of its argument attached to the result. For instance, ?5 produces an integer in the range [0..4] whereas ?-5 produces an integer in the range [-4..0]. The operand 0 is undefined for this operator. The target machine has support for this operation.

- There are two forms of iteration (loops) in C-. The **while** loops on a Boolean condition and the **for** loops on an integer index variable. The ID inside a **for** is its index variable, and is declared inside a new scope introduced by that **for** loop. The range of values of the integer index variable in a **for** loop are is defined by the optional keyword combinations of a **for** loop. These optional keyword combinations are an initial value, a range of values, or a range of values and a step indicator. The step indicator may be positive or negative. If a step indicator is not specified, it is assumed to be +1.

**C- Binary Operators**

| Operator | Operands | Return Type |
|---|---|---|
| and | bool, bool | bool |
| or | bool, bool | bool |
| == | equal types, arrays | bool |
| != | equal types, arrays | bool |
| < | equal types, arrays | bool |
| <= | equal types, arrays | bool |
| > | equal types, arrays | bool |
| >= | equal types, arrays | bool |
| = | equal types, arrays | type of lhs |
| += | int, int | int |
| -= | int, int | int |
| *= | int, int | int |
| /= | int, int | int |

| | | |
|---|---|---|
| + | int, int | int |
| - | int, int | int |
| * | int, int | int |
| / | int, int | int |
| % | int, int | int |
| [ ] | array, int | type of lhs |

**C- Unary Operators**

| Operator | Operands | Return Type |
|---|---|---|
| -- | int | int |
| ++ | int | int |
| not | bool | bool |
| * | array | int |
| - | int | int |
| ? | int | int |

**C- Array Operators**

| Operator | Type | Operands | Return Type |
|---|---|---|---|
| == | binary | equal types | bool |
| != | binary | equal types | bool |
| < | binary | equal (int or char) | bool |
| <= | binary | equal (int or char) | bool |
| > | binary | equal (int or char) | bool |
| >= | binary | equal (int or char) | bool |
| = | binary | equal types | type of lhs |
| [] | binary | array, int | type of lhs |
| * | unary | any | int |