C– Language Manual

Aditya Morolia; Roll No.: 20171177

September 4, 2020

This is a language manual for my programming language `C--`, a subset of the language C made to learn more about compilers and programming languages. This manual contains the micro and macro syntax of the language.

1 Remarks

- All `C--` keywords are lower case, and identifiers and keywords are case sensitive.
- Meta notations are references to describe the grammar.
- Micro syntax is described as a regex and the macro syntax is described using context free grammar.

1.1 Meta notations

`<foo>`	foo is a non-terminal symbol
foo or 'foo'	foo is a terminal symbol
[x]	0 or 1 occurrence of x
x*	zero or more occurrence of x
x^+,	comma separated list of one or more x
{ }	grouping
|	separate alternatives

2 Macro Syntax

⟨var_decl⟩ ⟶⟨type⟩ { ⟨id⟩
| ⟨id⟩ '=' ⟨literal⟩
| ⟨id⟩ '[' ⟨int_literal⟩ ']'
| ⟨id⟩ '[' ⟨int_literal⟩ '][' ⟨int_literal⟩ ']' }$^+$, ;

⟨method_decl⟩ ⟶{ ⟨type⟩ | **void** } ⟨id⟩ ([{ ⟨type⟩ ⟨id⟩ }$^+$,]) ⟨block⟩

⟨block⟩ ⟶'{' ⟨var_decl⟩* ⟨statement⟩* '}'

⟨type⟩ ⟶**int** | **bool** | **float**

⟨statement⟩ ⟶⟨location⟩ ⟨assign_op⟩ ⟨expr⟩ ;
| ⟨method_call⟩ ;
| **if** (⟨expr⟩) ⟨block⟩ [**else** ⟨block⟩]
| ⟨expr⟩ **?** ⟨statement⟩ **:** ⟨statement⟩ ;
| **while**(⟨expr⟩) ⟨block⟩
| **for** ([⟨id⟩=⟨expr⟩]; [⟨expr⟩]; [⟨expr⟩]) ⟨block⟩
| **return** [⟨expr⟩] ;
| **break** ;
| **continue** ;
| ⟨block⟩

⟨method_call⟩ ⟶**callout** (⟨string_literal⟩ [,⟨callout_arg⟩$^+$,])
| ⟨method_name⟩ ([⟨expr⟩$^+$,])

⟨location⟩ ⟶⟨id⟩
| ⟨id⟩‘[’ ⟨expr⟩ ‘]’
| ⟨id⟩‘[’ ⟨expr⟩ ‘][’ ⟨expr⟩ ‘]’

⟨expr⟩ ⟶⟨location⟩
| ⟨method_call⟩
| ⟨literal⟩
| ⟨expr⟩ ⟨bin_op⟩ ⟨expr⟩
| - ⟨expr⟩
| ! ⟨expr⟩
| (⟨expr⟩)

⟨callout_arg⟩ ⟶⟨expr⟩ | ⟨string_literal⟩

⟨bin_op⟩ ⟶⟨arith_op⟩ | ⟨rel_op⟩ | ⟨eq_op⟩ | ⟨cond_op⟩ | ⟨bool_op⟩

⟨literal⟩ ⟶⟨int_literal⟩ | ⟨char_literal⟩ | ⟨bool_literal⟩ | ⟨float_literal⟩

3 Micro Syntax

Token	RegEx
⟨string_literal⟩	"⟨char⟩*"
⟨char_literal⟩	'⟨char⟩'
⟨bool_literal⟩	**true** | **false**
⟨int_literal⟩	[+-]?[0-9]$^+$
⟨id⟩	[a-zA-Z][a-zA-Z0-9]*
⟨eq_op⟩	== | !=
⟨rel_op⟩	<= | >= | < | >
⟨arith_op⟩	+ | − | ∗ | / | %
⟨cond_op⟩	|| | &&
⟨assign_op⟩	+= | −= | =
⟨type⟩	**int** | **uint** | **bool** | **char**
⟨float_literal⟩	[+-]?([0-9]*[.])?[0-9]+
⟨bool_op⟩	& | | | !

4 Semantic Checks

A program can be completely correct syntactically but have no meaning semantically at all. Therefore, the compiler should run semantic checks on the source. Following semantic checks are bare minimum and should be performed:

1. **Undeclared Variable:** Check if all the identifiers/variables are declared before use in the same scope.
2. **Scope rules:** Check that every variable has been declared before it is used.
3. **Expression type mismatch:** Check if the type of the value finally assigned to a variable is the same as the variable type.
4. **Re-declaration of variables:** Check that no identifier has been redeclared in the current scope.
5. **Method signature and parameter type mismatch:** Check that every parameter passed in a method call corresponds to the type of the argument in the method signature.
6. **Misuse of keyword or reserved word:** Check that no reserved word has been used as an identifier.
7. **Array length:** Check if array ranges are within bounds and length is positive.