

A Grammar for the C- Programming Language (Version F09)

November 3, 2009

1 Introduction

This is a grammar for the C- programming language. This language is very similar to C and has a lot of features in common with a real-world structured programming language. There are also some real differences between C and C-. For instance the declaration of procedure arguments, allowable variable names, what constitutes the body of a procedure etc.

For the grammar that follows here are the types of the various elements by type font:

- **Keywords are in this type font.**
- **TOKEN CLASSES ARE IN THIS TYPE FONT.**
- *Nonterminals are in this type font.*

The symbol ϵ means the empty string.

1.1 Some Token Definitions

letter = a | ... | z | A | ... | Z

digit = 0 | ... | 9

quote = '

underbar = _

letdig = digit | letter | underbar

ID = letter⁺letdig*quote*

NUM = digit⁺

Also note that **white space** is ignored except that it must separate **ID**'s, **NUM**'s, and keywords.

Comments are treated like white space. Comments begin with // and run to the end of the line.

All **keywords** are in lowercase. You need not worry about being case independent since not all lex/flex programs make that easy.

2 The Grammar

1. $program \rightarrow declaration\text{-}list$
2. $declaration\text{-}list \rightarrow declaration\text{-}list\ declaration \mid declaration$
3. $declaration \rightarrow var\text{-}declaration \mid fun\text{-}declaration$
4. $var\text{-}declaration \rightarrow type\text{-}specifier\ var\text{-}decl\text{-}list ;$
5. $var\text{-}decl\text{-}list \rightarrow var\text{-}decl\text{-}list ,\ var\text{-}decl\text{-}id \mid var\text{-}decl\text{-}id$
6. $var\text{-}decl\text{-}id \rightarrow \mathbf{ID} \mid \mathbf{ID} [\mathbf{NUM}]$
7. $type\text{-}specifier \rightarrow \mathbf{int} \mid \mathbf{void} \mid \mathbf{bool}$
8. $fun\text{-}declaration \rightarrow type\text{-}specifier\ \mathbf{ID} (params)\ statement$
9. $params \rightarrow param\text{-}list \mid \epsilon$
10. $param\text{-}list \rightarrow param\text{-}list ;\ param\text{-}type\text{-}list \mid param\text{-}type\text{-}list$
11. $param\text{-}type\text{-}list \rightarrow type\text{-}specifier\ param\text{-}id\text{-}list$
12. $param\text{-}id\text{-}list \rightarrow param\text{-}id\text{-}list ,\ param\text{-}id \mid param\text{-}id$
13. $param\text{-}id \rightarrow \mathbf{ID} \mid \mathbf{ID} []$
14. $compound\text{-}stmt \rightarrow \{ local\text{-}declarations\ statement\text{-}list \}$
15. $local\text{-}declarations \rightarrow local\text{-}declarations\ var\text{-}declaration \mid \epsilon$
16. $statement\text{-}list \rightarrow statement\text{-}list\ statement \mid \epsilon$
17. $statement \rightarrow expression\text{-}stmt \mid compound\text{-}stmt \mid selection\text{-}stmt \mid iteration\text{-}stmt \mid return\text{-}stmt \mid break\text{-}stmt$
18. $expression\text{-}stmt \rightarrow expression ; \mid ;$
19. $selection\text{-}stmt \rightarrow \mathbf{if} (expression)\ statement \mid \mathbf{if} (expression)\ statement\ \mathbf{else}\ statement$
20. $iteration\text{-}stmt \rightarrow \mathbf{while} (expression)\ statement$
21. $return\text{-}stmt \rightarrow \mathbf{return} ; \mid \mathbf{return}\ expression ;$
22. $break\text{-}stmt \rightarrow \mathbf{break} ;$
23. $expression \rightarrow var = expression \mid var += expression \mid var -= expression \mid var ++ \mid var -- \mid simple\text{-}expression$

24. $var \rightarrow \mathbf{ID} \mid \mathbf{ID} [expression] \mid * \mathbf{ID}$
25. $simple-expression \rightarrow simple-expression \mathbf{or} and-expression \mid and-expression$
26. $and-expression \rightarrow and-expression \mathbf{and} unary-rel-expression \mid unary-rel-expression$
27. $unary-rel-expression \rightarrow \mathbf{not} unary-rel-expression \mid rel-expression$
28. $rel-expression \rightarrow sum-expression relop sum-expression \mid sum-expression$
29. $relop \rightarrow <= \mid < \mid > \mid >= \mid == \mid !=$
30. $sum-expression \rightarrow sum-expression sumop term \mid term$
31. $sumop \rightarrow + \mid -$
32. $term \rightarrow term mulop unary-expression \mid unary-expression$
33. $mulop \rightarrow * \mid / \mid \%$
34. $unary-expression \rightarrow unaryop unary-expression \mid factor$
35. $unaryop \rightarrow -$
36. $factor \rightarrow (expression) \mid var \mid call \mid constant$
37. $constant \rightarrow \mathbf{NUM} \mid \mathbf{true} \mid \mathbf{false}$
38. $call \rightarrow \mathbf{ID} (args)$
39. $args \rightarrow arg-list \mid \epsilon$
40. $arg-list \rightarrow arg-list , expression \mid expression$

3 Semantic Notes

- The only numbers are **ints**.
- There is no conversion or coercion between ints and bools or bools and ints.
- The unary asterisk is the only operator that takes an array as an argument. It takes an array and returns the size of the array.
- The boolean operators **and** and **or** use short cutting. For the **and** operator, if the left hand side is false the expression is false without evaluating the right hand side. Similarly with **or**, if the left hand side is true the right hand side is not evaluated.
- In if statements the **else** is associated with the most recent **if**.

- Expressions are evaluated in order consistent with operator associativity and precedence. No reordering allowed except in the case of shortcutting for boolean operators.
- Assignments in expressions happen at the time the assignment operator is encountered in the order of evaluation.
- Code that exits a procedure without a return returns a 0 for an int and **false** for a bool.

4 Example of C- Code

```
int ant(int bat, cat[]; bool dog, elk; int fox)
{
    int gnu, hog[100];

    hog = 3**cat;    // hog is 3 times the size of array passed to cat
    if (dog and elk or bat > cat[3]) dog = not dog;
    else fox++;
    if (bat <= fox) {
        while (dog) {
            int hog;
            hog = fox;
            dog = fred(fox++, cat)>641;
            if (hog>bat) break;
            fox += 7
        }
    }
    return (fox+bat*cat[bat])/-fox;
}
```

```
// note that functions are defined using a statement
int max(int a, b) if (a>b) return a; else return b;
```