A Grammar for the C- Programming Language

(version S07)

January 12, 2007

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

ID = letter$^+$digit*

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

25. *simple-expression* → *simple-expression* | *or-expression* | *or-expression*
26. *or-expression* → *or-expression* & *unary-rel-expression* | *unary-rel-expression*
27. *unary-rel-expression* → ! *unary-rel-expression* | *rel-expression*
28. *rel-expression* → *add-expression relop add-expression* | *add-expression*
29. *relop* → <= | < | > | >= | == | !=
30. *add-expression* → *add-expression addop term* | *term*
31. *addop* → + | −
32. *term* → *term mulop unary-expression* | *unary-expression*
33. *mulop* → * | / | %
34. *unary-expression* → - *unary-expression* | *factor*
35. *factor* → (*expression*) | *var* | *call* | *constant*
36. *constant* → **NUM** | **true** | **false**
37. *call* → **ID** (*args*)
38. *args* → *arg-list* | ϵ
39. *arg-list* → *arg-list* , *expression* | *expression*