Language Definition of C-

C- is a subset of the C programming language, called also C-Minus. The description for this language is largely due to Kenneth Louden's *Compiler Construction: Principles and Practice*. C- is C without pointers, with only integers, without some of the control constructs, and with no records.

Lexical Conventions of C-

1. Its keywords are as follows:
   a. **`else if int return void while`**
   b. All keywords are reserved, and must be written in lower case.
2. The following are special symbols in the language:
   a. **`+ - * / < <= > >= == != = ; , ( ) [ ] { }`**
   `/* */`
3. Other tokens are ***ID*** and ***NUM***, defined as follows
   a. an ***ID*** consists of a letter, followed by zero or more letters, digits, or underscores
   b. ***ID***s are case-sensitive
   c. a ***NUM*** is an integer literal (a digit followed by 0 or more digits)
4. White space consists of blanks, newlines, and tabs. White space must separate ***ID***s, ***NUM***s, and keywords; otherwise, it is ignored.
5. Comments are surrounded by **`/* */`**. Comments cannot be placed within tokens. They may not be nested.

C- Syntax

The grammar for C- may be specified as follows:

1. *program* → *declaration-list*
2. *declaration-list* → *declaration-list declaration* | *declaration*
3. *declaration* → *var-declaration* | *fun-declaration*
4. *var-declaration* → *type-specifier* ***ID*** **`;`** | *type-specifier* ***ID*** **`[`** ***NUM*** **`] ;`**
5. *type-specifier* → **`int`** | **`void`**
6. *fun-declaration* → *type-specifier* ***ID*** **`(`** *params* **`)`** *compound-stmt*
7. *params* → *param-list* | **`void`**
8. *param-list* → *param-list* **`,`** *param* | *param*
9. *param* → *type-specifier* ***ID*** | *type-specifier* ***ID*** **`[]`**
10. *compound-stmt* → **`{`** *local-declarations statement-list* **`}`**
11. *local-declarations* → *local-declarations var-declaration* | ε
12. *statement-list* → *statement-list statement* | ε

13. *statement* → *expression-stmt* | *compound-stmt* | *selection-stmt* | *iteration-stmt* | *return-stmt*
14. *expression-stmt* → *expression* **;** | **;**
15. *selection-stmt*→ **if (** *expression* **)** *statement* | **if (** *expression* **)** *statement* **else** *statement*
16. *iteration-stmt* → **while (** *expression* **)** *statement*
17. *return-stmt* → **return ;** | **return** *expression* **;**
18. *expression* → *var* **=** *expression* | *simple-expression*
19. *var* → ***ID*** | ***ID*** **[** *expression* **]**
20. *simple-expression* → *additive-expression relop additive-expression* | *additive-expression*
21. *relop* → **<=** | **<** | **>** | **>=** | **==** | **!=**
22. *additive-expression* → *additive-expression addop term* | *term*
23. *addop* → **+** | **-**
24. *term* → *term mulop factor* | *factor*
25. *mulop* → ***** | **/**
26. *factor* → **(** *expression* **)** | *var* | *call* | ***NUM***
27. *call* → ***ID*** **(** *args* **)**
28. *args* → *arg-list* | ε
29. *arg-list* → *arg-list* **,** *expression* | *expression*

Ecample C- Program

```
/* A C- program to compute gcd using Euclid's Algorithm. */

int gcd ( int u, int v )
{
  if ( v == 0 )
    return u ;
  else
    return gcd (v, u-u/v*v) ;
    /* u-u/v*v == u mod v */
}

void main(void)
{
  int x;
  int y;
  x = input();
  y = input();
  output(gcd(x,y));
}
```