Problem – Syntax-Directed Translation

Assume the following grammar of a simple, prefix expression language as introduced in homework 4, problem 3. The goal of this homework problem is to build a recursive descent LL(1) parser that generates ILOC code for this expression language.

Program ::= Stmtlist .
Stmtlist ::= Stmt NextStmt
NextStmt ::= ; Stmtlist | epsilon
Stmt ::= Assign | Print
Assign ::= ID = Expr
Print ::= ! ID
Expr ::= + Expr Expr |
       - Expr Expr |
       * Expr Expr |
       ID |
       ICONST
ID ::= a | b | c
ICONST ::= 1 | 2 | 3

Specify your parser in a pseudo language using the recursive descent parser design principles. There are several guidelines you should follow.

1. To generate code, call the pseudo utility function

   codeGen(InstructionType, op1, op2, op3).

   For example, the call codeGen_LoadI, 5, reg) generates a loadI instruction that loads the constant 5 into the register with the number “reg”, where reg is an integer valued variable. So if reg == 1, then the call will generate instruction loadI 5 ⇒ r1. Similarly, a call codeGen_Add, reg1, reg2, reg3) generates an add instruction for the two operand registers “reg1” and “reg2”, and the target register “reg3”. Again, reg1, reg2, and reg3 are integer valued variables.

2. Your code generator should not reuse registers as target registers of instructions, so we want to follow the register-register model. You should use another pseudo utility function, namely integer next_register(). A call to this function returns a “fresh” register number that has not been used before. The first call to next_register() returns the integer value 1.

3. r0 is our base register, and our “standard” r0 based memory layout should be supported. Your compiler should execute codeGen_LoadI, 1024, 0) before any other codeGen call.
4. There is no aliasing in our small expression language. Variables 'a', 'b', and 'c' should live at offsets 4, 8, and 12, respectively.

5. Important hint: Your recursive descent functions should return an integer value. This value should be the register number which is the final target register of the code generated by the call to the function.

6. Important hint: There are exceptions to the general design strategy of a recursive descent parser. You may need to “relax” this design principle a bit (“bend the rules”) to make things work. For example, how to deal with the function for nonterminal ID? What code should be generated?