<program> ::= <c> <public\_class> <c> | <c> <public\_class> <c> <program>

<public\_class> ::= <c> “public class” <capital\_identifier> “{“ <class\_block> “}” <c>

<class\_block> ::= <variable\_declarations> <method\_declarations> <nested\_class>

<nested\_class> ::= <c> <public\_class> <c> | “”

<variable\_declarations> ::= <c> <variable\_declaration> <c>

| <c> <variable\_declaration> <c> <variable\_declarations>

<variable declaration> ::= <variable\_modifier> <data\_type> <identifier> “;”

| <capital\_identifier> <identifier> “=” “new” <capital identifier> “(““)””;”

| “”

“(“ “)” “;”

<method\_declarations> ::= <c> <method\_declaration> <c> | <method\_declaration> <method\_declarations>

<method\_declaration> ::= <method\_modifier> <return\_type> <identifier> “(“ <parameters> “)” “{“ <method\_block> <return> “}”

<paramaters> ::= <data\_type> <identifier>

| <data\_type> <identifier> “,” <parameters>

| “”

<method\_block> = <c> <variable\_declarations> <c> <commands> <c>

<commands> ::= <c> <command> <c> | <c> <command> <c> <commands>

<commands> ::= <assignment> | <do\_loop> | <for\_loop> | <if\_else> | <switch\_case> | <print> | <break> | <method\_call> | “”

<command>

<assignment> ::= <identifier> “=” <expression> “;”

<expression> ::= <method\_call> | <literal> “;”| <object\_instance> | <operations> “;”| <member\_access>

<member\_access> ::= <identifier>”.”<identifier> “(”<arguments>”)” “;”

| <identifier> ”.” <identifier> “;”

<method\_call> ::= <identifier> “(”<arguments>”)” “;”

| <identifier>”.”<identifier> “(”<arguments>”)” “;”

<arguments> ::= <literal> | <literal> “,” <arguments> | <identifier> | <identifier> “,” <arguments>

<object\_instance> ::= “new” <capital identifier> “(“ “)” ”;”

<operations> ::= <literal> <operator\_symbol> <operations> | <literal>

<operator\_symbol> ::= “-” | “\*” | “/” | “+”

<do\_loop> ::= “do” “{“ <commands> ”}” “while” “(“<conditions>“)” “;”

<conditions> ::= <condition> | <condition> <logic\_operator> <conditions>

<condition> ::= <literal> <compare\_symbol> <identifier>

| <identifier> <compare\_symbol> <identifier>

| <literal> <compare\_symbol> <literal>

| <identifier> <compare\_symbol> <literal>

<compare\_symbol> ::= “>” | “<” | “==” | “!=”

<logic\_operators> ::= “&&” | “||”

<for\_loop> ::= “for” “(“ <for\_assignment> “;” <conditions> “;” <increment> “)” “{” <commands> “}”

<for\_assignment> ::= <identifier> “=” <literal> | <identifier> “=” <identifier>

<increment> ::= <identifier> “=” <identifier> <operator\_symbol> <literal>

| <identifier> “=” <identifier> <operator\_symbol> <identifier>

<if\_else> ::= “if” “(” <conditions> “)” “{” <commands> “}”

| “if” “(” <conditions> “)” “{” <commands> “}” “else” “{“ <commands> “}”

<switch\_case> ::= “switch” “(” <identifier> ”)” “{” <cases> “default:” <commands> “}”

|“switch” “(” <identifier> ”)” “{” <cases> “}”

<cases> ::= “case” <literal> “:” | “case” <literal> “:” <cases>

<print> ::= “out.print” “(“ <text> <variables> “)” “;”

<text> = <string\_literal>

<variables> ::= “,” <identifier> | “,” <identifier> <variables> | “”

<return> ::= “return” <literal> “;” | “return” <identifier> “;” | “return” <operations> “;”

| “return” <method\_call> | “return” <member\_access>

Returning a method call or a member access directly in a **return** statement is a feature found in languages that support object-oriented programming (OOP) concepts, such as classes and objects(C#, C++, Java). These constructs might be valid in these languages, but not in C.

<break> ::= “break” “;”

<variable\_modifier> ::= “public” | “private” | “”

<data\_type> ::= "int" | "char" | "double" | "boolean" | "String"

<return\_type> ::= <data\_type> | <void>

<literal> ::= <integer\_literal> | <char\_literal> | <double\_literal> | <boolean\_literal> | <string\_literal>

<integer\_literal> ::= <digit> | <digit> <integer\_literal>

<double\_literal> ::= <integer\_literal> “.” <integer\_literal> “d”

In general-purpose programming languages like C, C++, Java, Python, etc., the notation for double precision floating-point numbers typically does not include a suffix like "d". Instead, the presence of a decimal point or scientific notation (e.g., **3.14** or **1.5e3**) implies the use of double precision unless explicitly specified otherwise.

<boolean\_literal> ::= "true" | "false"

<string\_literal> ::= “**”**” <char> “**”**” | “**”**” <char> <string\_literal> “**”**”

<char\_literal> ::= “**‘**” <symbol> “**’**” | “**’**” <escape\_sequence> “**’**” | “**’**” <letter> ”**’**” | “’” <digit> “**’**”

<char> ::= <symbol> | <escape\_sequence> | <letter> | <digit>

<symbol> ::= '+' | '-' | '\*' | '/' | '%' | '=' | '&' | '|' | '^' | '~' | '!' | '<' | '>' | '?' | ':' | ';' | ',' | '.' | '(' | ')' | '[' | ']' | '{' | '}' | '#' | '\_' | '$' | '@' |

<escape\_sequence> ::= '\n' | '\t' | '\r' | '\\' | '\'' | '\"' | '\a' | '\b' | '\f' | '\v' | '\?' | '\0' | '\x'

<identifier> ::= <identifier\_head> <identifier\_body>

<identifier\_head> ::= “\_” | <letter>

<identifier\_body> ::= <identifier\_head> | <digit> | <identifier\_head> <identifier\_body>

| <digit> <identifier\_body> | “”

<capital\_identifier> ::= <capital\_letter> <identifier\_body>

<letter> ::= <lowercase\_letter> | <capital\_letter>

<lowercase\_letter> ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'

<capital\_letter> ::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'

<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

<c> ::= <comment> | “”

<comment> ::= <one\_line\_comment> | <multiple\_line\_comment>

<one\_line\_comment> :== “//” <string\_literal>

<multiple\_line\_comment> ::= “/\*” <string\_literal> “\*/”