```
program             ::= definition { definition }
definition          ::= dataDefinition | functionDefinition
dataDefinition      ::= INT declarator { "," declarator } ";"
declarator          ::= ID
functionDefinition  ::= [INT] functionHeader functionBody
functionHeader      ::= declarator parameterList
parameterList       ::= "(" [ identifierList] ")" { parameterDeclaration }
identifierList      ::= ID { "," ID }
parameterDeclaration ::= INT declarator { "," declarator } ";"
functionBody        ::= "{" {dataDefinition} { statement }  "}"
statement           ::= [ expression ] ";"
                      | "{" {dataDefinition} { statement }  "}"
                      | IF "(" expression ")" statement [ELSE statement]
                      | WHILE "(" expression ")" statement
                      | BREAK ";"
                      | CONTINUE ";"
                      | RETURN [ expression ] ';'
expression          ::= binary { "," binary }
binary              ::=    ID        "="       binary
                      |    ID        "+="      binary
                      |    ID        "-="      binary
                      |    ID        "*="      binary
                      |    ID        "/="      binary
                      |    ID        "%="      binary
                      |    binary    "+"       binary
                      |    binary    "-"       binary
                      |    binary     "*"      binary
                      |    binary    "/"       binary
                      |    binary    "%"       binary
                      |    binary    "&"       binary
                      |    binary    "|"       binary
                      |    binary    "^"       binary
                      |    binary    "<"       binary
                      |    binary    ">"       binary
                      |    binary    ">="      binary
                      |    binary    "<="      binary
                      |    binary    "=="      binary
                      |    binary    "!="      binary
                      |    unary
unary               ::= "++" ID
                      | "--" ID
                      | primary
primary                    ::= ID
                      | CONST
                      | "(" expression ")"
                      | ID "(" [argumentList] ")"
argumentList        ::= binary { "," binary }
```