Format générale

```
using System;
namespace YourNamespace
{
 class YourMainClass

    {
        Type_return NameMethod( arguments_method ){
            //Your mrthod starts here...
        }

        .
        .
        .

        static void Main(string[] args)
        {
            //Your program starts here...
        }
    }
}
```

Start

Start : {using name_space;} program

name_space : System

program : namespace identifier{name_space_implement} | name_space_implement

Name_space_implement : [p_p] *class* identifier{ class_implement }

p_p : private | public

class_implement : {methode_declaration} [main_implement]

main_implement : [public] static void Main(string[] identifier) methode_body

methode_declaration :

[p_p] [static] return_type methode_name([formal_parameter_list]) methode_body

return_type : type | void

methode_name : meth_identifier

formal_parameter_list : fixed-parameters {,parameter-array} | parameter-array

fixed-parameters: fixed-parameter {, fixed-parameter}

fixed-parameter : type identifier

parameter-array: params array-type identifier

methode_body: block | ;

Types

type: value-type [[]]

value-type: int | long |char | float | double | bool

array-type: value-type[]

identifier : letter {character}

Statement

block: { {statement} }

statement: declaration-statement | embedded-statement

embedded-statement: block | print; | statement-expression; | selection-statement | iteration-statement | jump-statement | try-statement

print: [System.]Console.Writeline({expression+} [expression])

declaration-statement: local-variable-declaration ; | local-constant-declaration ;

local-variable-declaration: type variable-declarator {, variable-declarator}

variable-declarator: identifier [= variable-initializer]

variable-initializer: expression

local-constant-declaration: const type constant-declarator {, constant-declarator}

constant-declarator: identifier = expression

statement-expression: invocation-expression | assignment

selection-statement: if (boolean-expression) embedded-statement [else embedded-statement] | switch (identifier) { { switch-section } }

switch-section: { switch-label } { statement } [break;]

switch-label: case expression : | default :

iteration-statement: while (boolean-expression) embedded-statement | do embedded-statement while (boolean-expression) ; | for ([for-initializer] ; [boolean-expression] ; [for-iterator]) embedded-statement | foreach (type identifier in identifier) embedded-statement

for-initializer: local-variable-declaration | assignment

for-iterator: statement-expression

jump-statement: break ; | continue ; | return [expression] ;

boolean-expression: expression

Expression

expression: conditional-or-expression

assignment: identifier assignment-body

assignment-body: ++ | -- | assignment-operator (type) expression

assignment-operator: = | += | -= | *= | /= | %= | ^=

conditional-or-expression: conditional-and-expression { || conditional-and-expression}

conditional-and-expression: inclusive-or-expression { && inclusive-or-expression}

inclusive-or-expression: relational-expression {equality-operator relational-expression}

equality-operator: == | !=

relational-expression: additive-expression {relational-operator additive-expression }

relational-operator: < | > | <= | >=

additive-expression: multiplicative-expression {additive-operator multiplicative-expression}

additive-operator: + | -

multiplicative-expression: unary-expression { multiplicative-operator primary-expression }

multiplicative-operator: * | / | %

primary-expression: identifier [[inum]] | (conditional-or-expression)| invocation-expression | inum | fnum | true | false | string | ‘character’ | array-creation-expression

invocation-expression: meth_identifier ([argument-list])

argument-list: argument {, argument}

argument: expression

array-creation-expression: new value-type [array-length [array-initializer]

array-length: inum] |]

array-initializer: {{expression,} [expression]}