INScore v.1.27
-
Scripting language

D. Fober

Centre national de création musicale

Contents

.

Chapter 1

INScore Scripting Language

1.1 Introduction

INScore scripting language is based on a textual version of OSC messages, extended with variables, Javascript sections and Mathematical expressions. INScore scripts files are expected to carry a `.inscore` extension. You can drop them to the INScoreViewer application or to any opened INScore scene.

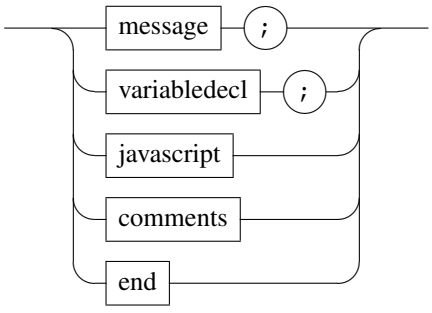
The application or scene state can be saved (using the `save` message) as files containing textual OSC messages. These files can be edited or created from scratch using any text editor.

1.2 Statements

An INScore file is a list of textual expressions. An expression is:

- a message: basically a textual OSC message extended to support URL like addresses and variables as parameters.
- a variable declaration.
- a javascript section that may generate messages as output.
- comments.
- an end marker '`__END__`' to declare a script end. After the marker, the remaining part of the script will be ignored.

expression



Messages and variables declarations must be followed by a semicolon that is used as statements separator.

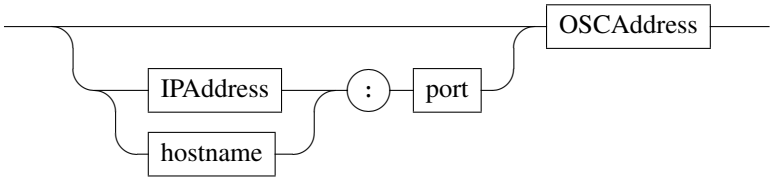
1.3 Messages

Messages are basic OSC messages that support an OSC address extension scheme and relative addresses that are described below. Messages parameters can be replaced by variables that are evaluated at parsing level. Variables are described in section 1.5.

1.3.1 Extended OSC addresses

OSC addresses can be extended to target other applications, including on other hosts.

address



Using the address extension scheme, a script may be designed to initialize an INScore scene and external applications as well, including on remote hosts.

EXAMPLE

Initializing a score and an external application listening on port 12000 and running on a remote host named `host.adomain.net`.

```
/ITL/scene/score set gmnf 'myscore.gmn';
host.adomain.net:12000/run 1;
```

1.3.2 Relative addresses

Relative addresses have been introduced to provide more flexibility in the score design process. A relative address starts with `'./'`. It is evaluated in the context of the message receiver: a legal OSC address is dynamically constructed using the receiver address to prefix the relative address.

EXAMPLE

```
the relative address   ./score
addressed to           /ITL/scene/layer
will be evaluted as    /ITL/scene/layer/score
```

The receiver context may be:

- the INScore application address (i.e. `/ITL`) for messages enclosed in a file loaded at application level (using the `load` message addressed to the application) or for files dropped to the application or given as arguments of the INScoreViewer application.
- a scene address for messages enclosed in a file loaded at scene level (using the `load` message addressed to a scene) or for files or messages dropped to a scene window.
- any object address when the messages are passed as arguments of an `eval` message (see OSCMsg reference).

EXAMPLE

Using a set of messages in different contexts:

```
score = (
   ./score set gmn '[a f g]',
   ./score scale 2.
);
/ITL/scene/l1 eval $score;
/ITL/scene/l2 eval $score;
```

NOTE

Legal OSC addresses (i.e. addresses that start with '/') that are given as argument of an `eval` message are not affected by the address evaluation.

1.4 Types

The message parameters types are constrained by the OSC protocol: any parameter is converted to an OSC type (i.e. int32, float or string) at parsing level. A special attention must be given to strings in order to discriminate addresses and parameters. Strings intended as parameters must:

- be quoted, using single or double quotes. Note that an ambiguous quote included in a string can be escaped using a '\'.
- or avoid any special characters i.e. any other character than `[_-a-zA-Z0-9]`.

NOTE

`text` objects are permissive with the above rules: spaces don't have to be quoted, they accept also numbers as input arguments (they are converted to strings).

EXAMPLE

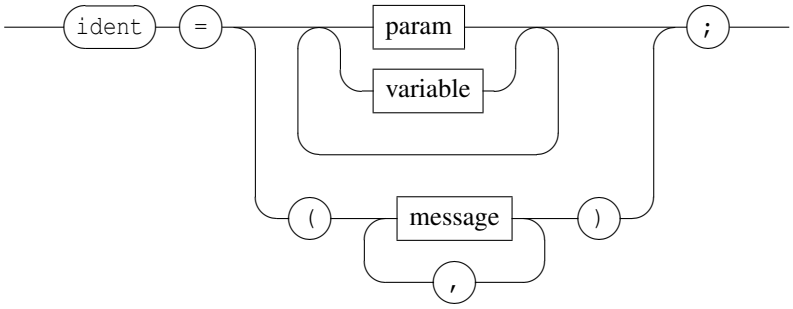
Different string parameters

```
/ITL/scene/text set txt "Hello world"; #string including a space can be quoted
/ITL/scene/text set txt Hello world; #text objects support stream like parameters
/ITL/scene/img set file 'anImage.png'; #dots must be quoted too
/ITL/scene/foo set txt no_quotes_needed;
```

1.5 Variables

A variable declaration associates a name with a list of parameters or a list of messages. Parameters must follow the rules given in section 1.4. They may include previously declared variables. A message list must be enclosed in parenthesis and a comma must be used as messages separator.

variabledecl



EXAMPLE

Variables declarations

```
color = 200 200 200;
colorwithalpha = $color 100; #using another variable
msgsvar= (                   #a variable referring to a message list
       localhost:7001/world "Hello world",
       localhost:7001/world "how are you ?" );
```

A variable may be used in place of any message parameter. A reference to a variable must have the form `$ident` where `ident` is a previously declared variable. A variable is evaluated at parsing level and replaced by its content.

EXAMPLE

Using a variable to share a common position:

```
x = 0.5;
/ITL/scene/a x $x;
/ITL/scene/b x $x;
```

Variables can be used in interaction messages as well, which may also use the variables available from events context. To differentiate between a *script* and an *interaction* variable, the latter must be quoted to be passed as strings and to prevent their evaluation by the parser.

EXAMPLE

Using variables in interaction messages: $sx is evaluated at event occurrence and $y is evaluated at parsing level.

```
y = 0.5;
/ITL/scene/foo watch mouseDown (/ITL/scene/foo "$sx" $y);
```

1.6 Environnement variables

Environnement variables are predefined variables available in a script context. They provide information related to the current context. Current environment variables are:

- **`OSName`**: gives the current operating system name. The value is among `"MacOS"`, `"Windows"`, `"Linux"`, `"Android"`, `"iOS"` and `"Web"`.
- **`OSId`** : gives the current operating system as a numeric identifier. Returned value is (in alphabetic order):
  - 1 for Android
  - 2 for iOS
  - 3 for Linux
  - 4 for MacOS
  - 5 for Windows
  - 6 for the Web environment

NOTE

There is nothing to prevent overriding of an environment variable. It's the script responsibility to handle variable names collisions.

1.7 Message based parameters

A message parameter may also use the result of a `get` message as parameter specified like a message based variable. The message must be enclosed in parenthesis with a leading $ sign.

msgparam

EXAMPLE

Displaying INScore version using a message parameter:

```
/ITL/scene/version set txt "INScore version is" $(/ITL get version);
```

NOTE

Message based parameters are evaluated by the parser. Thus when the system state is modified by a script before a message parameter, these modifications won't be visible at the time of the parameter evaluation because all the messages will be processed by the next time task. For example:

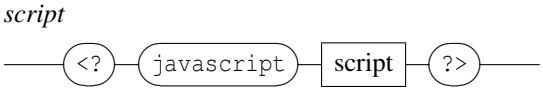
```
/ITL/scene/obj x 0.1;
/ITL/scene/foo x $(/ITL/scene/foo get x);
```

x position of `/ITL/scene/foo` will be set to x position of `/ITL/scene/obj` at the time of the script evaluation (that may be different to 0.1).

1.8 Javascript

INScore supports Javascript as scripting languages. Javascript is embedded using the Qt Javascript engine. A script section is indicated similarly to a Javascript section in html i.e. enclosed in an opening `<?` and a closing `?>`.

script



The principle of using Javascript sections in INScore files is the following: the Javascript sections are passed to the Javascript engine and are expected to produce textual INScore messages on output. These messages are then parsed as if replacing the corresponding script section.
INScore variables are exported to the Javascript environment.

NOTE

A Javascript section may produce not output, for example when it declares functions to be used later.

EXAMPLE

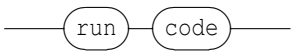
```
<?  javascript
   "/ITL/scene/version set txt 'Javascript v.  " + version() + "';"
?>
```

A single persistent Javascript context is created at application level and shared with each scene.

1.8.1 The Javascript object

The Javascript engine is available at runtime at the address `/ITL/`*`scene`*`/javascript`. It has a `run` method that takes a javascript string as parameter.

javascript



The `run` method evaluates the code. Similarly to javascript sections in scripts, the output of the evaluation is expected to be a string containing valid INScore messages that are next executed. Actually, including a javascript section in a script is equivalent to send the `run` message with the same code as parameter to the javascript object.

The Javascript engine is based on the Qt5 Javascrip engine, extended with INScore specific functions:

- **`version()`**: gives the javascript engine version number as a string.
- **`print(val1 [, val2 [, ...]])`**: print the arguments to the OSC standard output. The arguments list is prefixed by 'javascript:'. The function is provided for debug purpose.
- **`readfile(file)`**: read a file and returns its content as a string. The file name could be specified as an absolute or relative path. When relative, the file is searched in the application current `rootPath`.
- **`post(address [,...])`**: build an OSC message and post it for delayed processing i.e. to be processed by the next time task. `address` is an OSC or an extended OSC address. Optional arguments are the message parameters.
- **`osname()`**: gives the current operating system name.
- **`osid()`**: gives the current operating system as a numeric identifiant. Returned value is (in alphabetic order):
  - 1 for Android
  - 2 for iOS.
  - 3 for Linux,
  - 4 for MacOS,
  - 5 for Windows

EXAMPLE

Posting a message from a Javascript section:

```
<?javascript
   post ("/ITL/scene/obj", "dalpha", -1);";
   // The message /ITL/scene/obj dalpha -1
   // will be evaluated by the next time task.
?>
```

Declaration of a Javascript function to be used later:

```
<?javascript
   // declare a function foo()
   function foo(arg) {
     return "/ITL/scene/obj set txt foo called with " + arg + ";";
   }
?>

# call the foo function
<?javascript foo(1)?>

# or call the foo function using the run message
/ITL/scene/javascript run "foo(1)";
```

1.9 Comments

There are two ways to comment code inside a script:

- line comments: the '#' character is used for line comments, anything after a '#' is ignored.
- section comments: section comments start with '(*' and end with '*)', anything between them is ignored.

EXAMPLE

```
# this is a line comment
/ITL/scene/* del;

(*
   here is a section comment
   section comments are multi-lines comments
*)

/ITL/scene/hello set txt "Hello world!"; # the preceding message is not commented

__END__

and everything below the __END__ marker is also ignored
```

Chapter 2

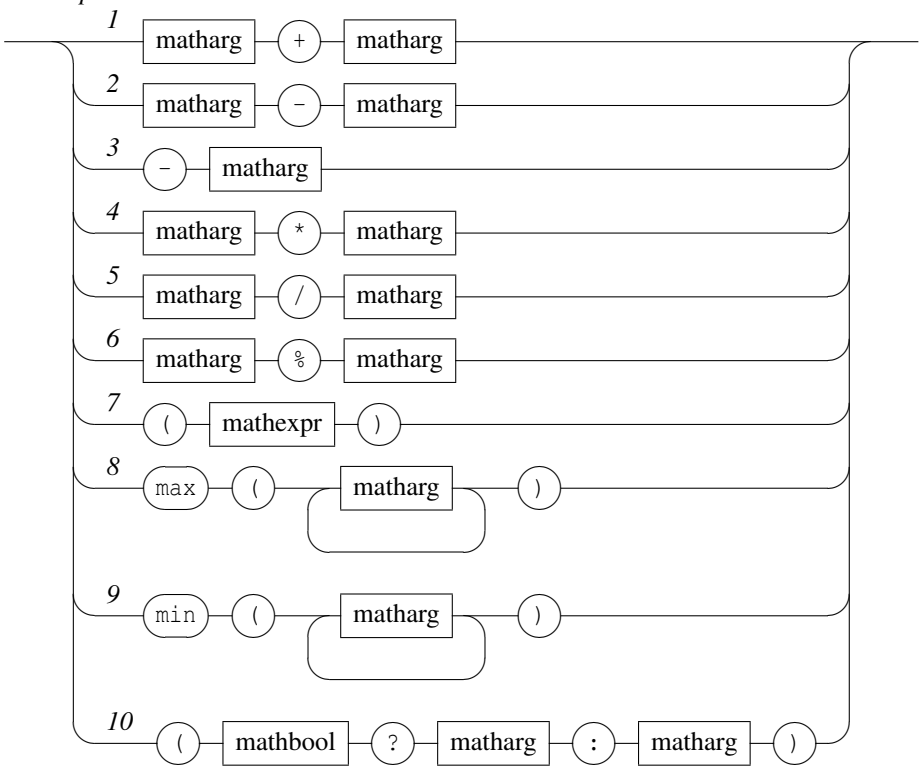
Mathematical expressions

Since INScore version 1.20, mathematical expressions have been introduced as messages arguments. These expressions allows to compute values at parsing time.

2.1 Operators

Basic mathematical expressions are supported. They are listed below.

mathexpr



Note that *matharg* could be a *mathexpr* as well (see section 2.2).

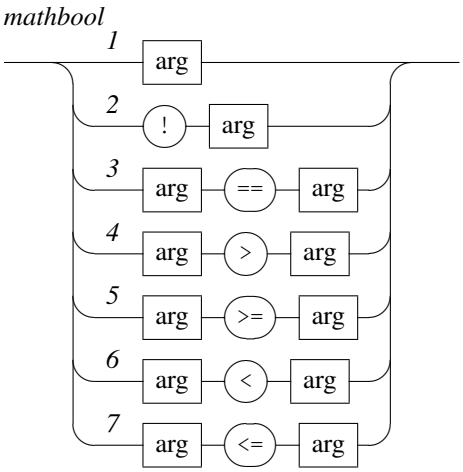
- **1**: addition.
- **2**: substraction.
- **3**: negation.
- **4**: multiplication.
- **5**: division.
- **6**: modulo.
- **7**: parenthesis, to be used for evaluation order.

- **8**: gives the maximum value.
- **9**: gives the minimum value.
- **10**: conditional form: returns the first `matharg` if `mathbool` is true, otherwise returns the second one.

EXAMPLE
Some simple mathematical expressions used as message parameters:

```
/ITL/scene/myObject x 0.5 * 0.2;
/ITL/scene/myObject y ($var ?  1 :  -1);
```

Boolean operations are the following:



- **1**: evaluate the argument as a boolean value.
- **2**: evaluate the argument as a boolean value and negates the result.
- **3**: check if the arguments are equal.
- **4**: check if the first argument is greater than the second one.
- **5**: check if the first argument is greater or equel to the second one.
- **6**: check if the first argument is less than the second one.
- **7**: check if the first argument is less or equal to the second one.

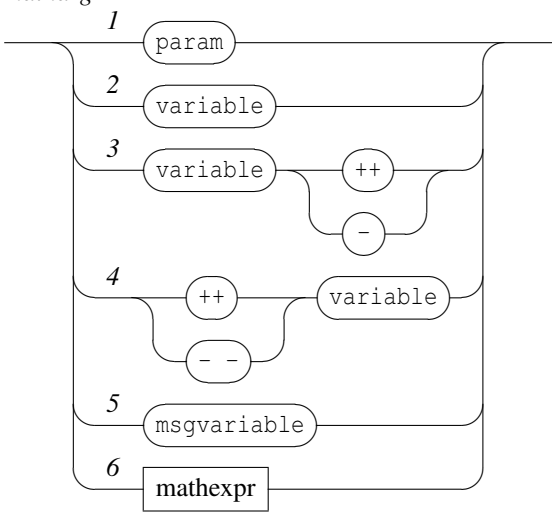
EXAMPLE
Compare two variables:

```
/ITL/scene/myObject x ($var1 > $var2 ?  1 :  -1);
```

2.2 Arguments

Arguments of mathematical operations are the following:

matharg



- **1**: any message parameter.
- **2**: a variable value.
- **3**: a variable that is post incremented or post decremented.
- **4**: a variable that is pre incremented or pre decremented.
- **5**: a message based variable.
- **6**: a mathematical expression.

2.3 Polymorphism

Since INScore's parameters are polymorphic, the semantic of the operations are to be defined notably when applied to non numeric arguments. Actually, a basic OSC message parameter type is between `int32`, `float32` and `string`. However, due to the extension of the scripting language, parameters could also be arrays of any type, including mixed types (e.g. resulting from variable declarations).

2.3.1 Numeric values

For numeric arguments, automatic type conversion is applied with a precedence of `float32` i.e. when one of the argument's type is `float32`, the result is also `float32` (see Table 2.3.1).

arg1	arg2	output
int32	int32	int32
float32	int32	float32
int32	float32	float32
float32	float32	float32

Table 2.1: Numeric operations output

2.3.2 Strings

For `string` parameters, operations that have an obvious semantic (like + applied to two strings) are defined (see Table 2.3.2) , the others are undefined and generate an error (see Table 2.3.2).

Boolean operations are supported on `string` only when both arguments are strings (see Table 2.3.2). Other types combination generate an error.

operation	evaluates to	comment
`string` + `string`	`string`	concatenate the two strings
`string` + `num`	`string` + *string*(`num`)	`num` is converted to string
`num` + `string`	*string*(`num`) + `string`	"
@max(`string string`)	`string`	select the largest string
@min(`string string`)	`string`	select the smallest string

Table 2.2: Supported operations on strings

operation	comment
`string` *op* `string`	where *op* is in [- * / %]
`string` *op* `num`	"
`num` *op* `string`	"
`-string`	
prefixed or postfixed `string`	

Table 2.3: Non supported operations on strings

2.3.3 Arrays

For arrays, the operation is distributed inside the array elements:

$$arg\ op\ [v_1\ ...\ v_n] := [arg\ op\ v_1\ ...\ arg\ op\ v_n]$$

or

$$[v_1\ ...\ v_n]\ op\ arg := [v_1\ op\ arg\ ...\ v_n\ op\ arg]$$

When both parameters are arrays, the operation is distributed from one array elements to the other array elements when the arrays have the same size and it generates an error when the sizes differ:

$$[a_1\ ...\ a_i]\ op\ [b_1\ ...\ b_i] := [a_1\ op\ b_1\ ...\ a_i\ op\ b_i\ ]$$

Boolean operations on arrays are evaluated as the logical *and* of it's element's boolean values and generate an error when the arrays sizes differ.

boolean operation	evaluated as
`string` == `string`	regular strings comparison
`string` > `string`	alphabetical strings comparison
`string` >= `string`	"
`string` < `string`	"
`string` <= `string`	"

Table 2.4: Supported boolean operations on strings

Chapter 3

Appendices

3.1 Grammar definition

```
//_____________________________________________________
// relaxed simple ITL format specification
//_____________________________________________________
start        : expr
             | start expr
             ;
//_____________________________________________________
// expression of the script language
//_____________________________________________________
expr         : message  ENDEXPR
             | variabledecl ENDEXPR
             | script
             | ENDSCRIPT
             ;
//_____________________________________________________
// javascript and support
//_____________________________________________________
script         : JSCRIPT
             ;
//_____________________________________________________
// messages specification (extends osc spec.)
//_____________________________________________________
message        : address
             | address params
             | address eval LEFTPAR messagelist RIGHTPAR
             | address eval variable
             ;
messagelist : message
             | messagelist messagelistseparator message
             ;
messagelistseparator    : COMMA
                        | COLON
                        ;
//_____________________________________________________
// address specification (extends osc spec.)
address        : oscaddress
             | relativeaddress
             | urlprefix oscaddress
             ;
oscaddress   : OSCADDRESS
             ;
```

```
relativeaddress    : POINT oscaddress
            ;
urlprefix    : hostname  UINT
            | STRING COLON UINT
            | IPNUM COLON UINT
            ;
hostname    : HOSTNAME
            ;
identifier    : IDENTIFIER
            | HOSTNAME
            | REGEXP
            ;
//_____________________________________________
// parameters definitions
// eval need a special case since messages are expected as argument
eval        : EVAL
            | variable
            | params variable
            | params param
            ;
params        : sparam
            | params sparam
            | mathexpr
            | params mathexpr
            ;
variable    : VARIABLE
            | VARIABLEPOSTINC
            | VARIABLEPOSTDEC
            | VARIABLEPREINC
            | VARIABLEPREDEC
            ;
msgvariable    : VARSTART LEFTPAR message RIGHTPAR
            ;
param    : number
        | FLOAT
        | identifier
        | STRING
        ;
sparam    : expression
        | LEFTPAR messagelist RIGHTPAR
        | script
        ;
//_____________________________________________
// math expressions
mathexpr    : param
            | variable
            | msgvariable
            | mathexpr ADD mathexpr
            | mathexpr SUB mathexpr
            | MINUS mathexpr
            | mathexpr MULT mathexpr
            | mathexpr DIV mathexpr
            | mathexpr MODULO mathexpr
            | LEFTPAR mathexpr RIGHTPAR
            | MIN LEFTPAR mathmin RIGHTPAR
            | MAX LEFTPAR mathmax RIGHTPAR
            | LEFTPAR mathbool QUEST mathexpr COLON mathexpr RIGHTPAR
            ;
mathmin     : mathexpr
```

```
            | mathmin mathexpr
            ;
mathmax     : mathexpr
            | mathmax mathexpr
            ;
mathbool     : mathexpr
            | NEG mathexpr
            | mathexpr EQ mathexpr
            | mathexpr NEQ mathexpr
            | mathexpr GREATER mathexpr
            | mathexpr GREATEREQ mathexpr
            | mathexpr LESS mathexpr
            | mathexpr LESSEQ mathexpr
            ;
//_____________________________________________________
// variable declaration
variabledecl : varname EQUAL params
            ;
varname        : IDENTIFIER
            ;
//_____________________________________________________
// misc
number        : UINT
            | INT
            ;
//_____________________________________________________
// expression declaration
expression    : EXPRESSION
            ;
```

3.2 Lexical tokens

```
//_____________________________________________________
// numbers
//_____________________________________________________
INT       a signed integer
UINT      an unsigned integer
FLOAT     a floating point number

//_____________________________________________________
// hosts addresses
//_____________________________________________________
// allowed character set for host names (see RFC952 and RFC1123)
HOSTNAME       : [-a-zA-Z0-9]+
IPNUM          :  {DIGIT}+"."{DIGIT}+"."{DIGIT}+"."{DIGIT}+

//_____________________________________________________
// OSC addresses
//_____________________________________________________
// allowed characters for identifiers
IDENTIFIER     : [_a-zA-Z][_a-zA-Z0-9]*
REGEXP           see OSC doc for regular expressions
OSCADDRESS

//_____________________________________________________
// strings
//_____________________________________________________
STRING          : ("/"|("."".""?"/")*)([^ \t\\/?:*><|"';=]+"/"?)+"."[_a-zA-Z0-9]+
```

```
                or quoted strings that can include any character
                quotes could be single (') or double quotes (")

//_____________________________________________________
// scripting
//_____________________________________________________
JSCRIPT        : <?javascript any javascript code ?>
VARIABLE       : the name of a name

//_____________________________________________________
// misc.
//_____________________________________________________
POINT          : '.'
VARSTART       : '$'
COLON          : ':'
COMMA          : ','
LEFTPAR        : '('
RIGHTPAR       : ')'
EQUAL          : '='
ENDEXPR        : ';'
ENDSCRIPT      : "__END__"
EVAL           : "eval"

//_____________________________________________________
// score expressions
//_____________________________________________________
EXPRESSION    expr( a valid score expression )
              see 'Score expressions grammar'

//_____________________________________________________
// math expressions
//_____________________________________________________
ADD            : '+'
DIV            : '/'
EQ             : '=='
GREATER        : '>'
GREATEREQ      : '>='
LESS           : '<'
LESSEQ         : '<='
MINUS          : '-'
MODULO         : '%'
MULT           : '*'
NEG            : '!'
QUEST          : ' ? '
SUB            : '- '
MAX            : '@max'
MIN            : '@min'

VARIABLEPOSTDEC : $VAR--
VARIABLEPOSTINC : $VAR++
VARIABLEPREDEC  : --$VAR
VARIABLEPREINC  : ++$VAR
```