

GTL Template language

Jean-Luc Béchenec

July 25, 2016

Contents

1	Data types	1
1.1	getters applicable to any data type	2
1.2	int data type	2
1.2.1	int operators	2
1.2.2	int getters	3
1.2.3	int setters	4
1.3	The float data type	5
1.3.1	float operators	5
1.3.2	float getters	5
1.4	The string data type	6
1.4.1	string operators	6
1.4.2	string getters	7
1.5	The bool data type	9
1.5.1	bool operators	9
1.5.2	bool getters	10

1 Data types

GTL supports the following data types:

int arbitrary precision integer numbers. The GMP library is used;

float 64 bits floating point numbers;

bool standard boolean;

type the type of a data;

string unicode strings;

struct structured data;

list lists of data, may be accessed as a table;

map map (aka dictionary) of data.

Each type has its set of operators, getters and setters. The expression, for getters, or the variable, for setters, is called the *target*. Getters return a value related to the data but do not change it. They are used to get information from a data or to convert it into another type. Setter may target a literal expressions. Setters change the data and do not return anything. Setter may only target a variable. Getters and setters may have arguments. Syntax for getters without argument is as follow:

`[expression getter]`

When the getter takes arguments, they are listed after a colon and separated commas as follow:

`[expression getter : arg1, arg2, ..., argN]`

Syntax for setters without arguments is as follow:

`[!variable setter]`

When the setter takes arguments, they are listed after a colon and separated by commas as follow:

`[!variable setter : arg1, arg2, ..., argN]`

1.1 getters applicable to any data type

type returns the data type of the expression. See the section ??.

isANumber returns true if the expression is a number: int or float, false otherwise.

1.2 int data type

The int data type support arbitrary precision arithmetic by using the GNU Multiple Precision Arithmetic Library (GMP).

1.2.1 int operators

The int datatype supports the following operators:

Unary operators

Operator	Expression type	Meaning
+	<code>int ← +int</code>	Plus operator. No effect
-	<code>int ← -int</code>	Minus operator. Negation
~	<code>int ← ~int</code>	Not operator. Complementation by 1

Binary arithmetic operators

Operator	Expression type	Meaning
+	<code>int ← int + int</code>	Addition
-	<code>int ← int - int</code>	Substraction
*	<code>int ← int * int</code>	Multiplication
/	<code>int ← int / int</code>	Division
mod	<code>int ← int mod int</code>	Modulus

Binary bitwise operators

Operator	Expression type	Meaning
&	<code>int ← int & int</code>	bitwise and
	<code>int ← int int</code>	bitwise or
^	<code>int ← int ^ int</code>	bitwise exclusive or
<<	<code>int ← int << int</code>	shift left
>>	<code>int ← int >> int</code>	shift right

Comparison operators

Operator	Expression type	Meaning
!=	<code>bool ← int != int</code>	Not equal
==	<code>bool ← int == int</code>	Equal
>	<code>bool ← int > int</code>	Greater than
<	<code>bool ← int < int</code>	Lower than
>=	<code>bool ← int >= int</code>	Greater or equal
<=	<code>bool ← int <= int</code>	Lower or equal

1.2.2 int getters

getter	Type	Meaning
string	string	Returns a string decimal representation of the int expression. <code>[42 string]</code> returns string "42".
hexString	string	Returns a string hexadecimal representation of the int expression prefixed by 0x. If the expression is negative a '-' is inserted before. <code>[42 hexString]</code> returns string "0x2A". <code>[-1 hexString]</code> returns string "-0x1".
xString	string	Returns a string hexadecimal representation of the int expression. If the expression is negative a '-' is inserted before. <code>[42 xString]</code> returns string "2A". <code>[-42 xString]</code> returns string "-2A".

getter	Type	Meaning
numberOfBytes	int	Returns the number of bytes needed to store an unsigned expression. [255 numberOfBytes] returns 1, [256 numberOfBytes] returns 2
signedNumberOfBytes	int	Returns the number of bytes needed to store a signed expression. [127 numberOfBytes] returns 1, [128 numberOfBytes] returns 2
numberOfBits	int	Returns the number of bits needed to store an unsigned expression. [63 numberOfBits] returns 6, [64 numberOfBits] returns 7
signedNumberOfBits	int	Returns the number of bits needed to store a signed expression. [63 signedNumberOfBits] returns 7, [64 signedNumberOfBits] returns 8
sign	int	Returns -1 if the expression is strictly negative, 0 if it is null and +1 if the expression is strictly positive.
fitsUnsignedInByte	bool	Returns true if the expression fits in an unsigned byte, false otherwise.
fitsSignedInByte	bool	Returns true if the expression fits in a signed byte, false otherwise.
fitsUnsignedInWord	bool	Returns true if the expression fits in an unsigned 16 bits word, false otherwise.
fitsSignedInWord	bool	Returns true if the expression fits in a signed 16 bits word, false otherwise.
fitsUnsignedInLong	bool	Returns true if the expression fits in an unsigned 32 bits long, false otherwise.
fitsSignedInLong	bool	Returns true if the expression fits in a signed 32 bits long, false otherwise.
fitsUnsignedInLongLong	bool	Returns true if the expression fits in an unsigned 64 bits long long, false otherwise.
fitsSignedInLongLong	bool	Returns true if the expression fits in a signed 64 bits long long, false otherwise.
abs	int	Returns the absolute value of the expression.
bitAtIndex	bool	This getter takes one argument: index. It returns true if the bit at index index is set and false otherwise. index 0 corresponds to the lowest significant bit. [1 bitAtIndex: 0] returns true

1.2.3 int setters

setter	Meaning
setBitAtIndex	This setter takes two arguments. The first one, <code>value</code> , is a bool. The second one, <code>index</code> , is the index of the bit to set. if <code>value</code> is <code>true</code> the bit is set to 1 and to 0 otherwise. Assuming <code>a</code> contains 0 at start, <code>[!a setBitAtIndex: true, 0]</code> sets <code>a</code> to 1.
complementBitAtIndex	This setter takes one argument, <code>index</code> , which is the index of the bit to complement. Assuming <code>a</code> contains 1 at start, <code>[!a complementBitAtIndex: 1]</code> sets <code>a</code> to 3.

1.3 The float data type

The float data type is the standard IEEE754 64 bits floating point number.

1.3.1 float operators

The float data type supports the following operators:

Unary operators

Operator	Expression type	Meaning
+	<code>float ← +float</code>	Plus operator. No effect
-	<code>float ← -float</code>	Minus operator. Negation

Binary arithmetic operators

Operator	Expression type	Meaning
+	<code>float ← float + float</code>	Addition
-	<code>float ← float - float</code>	Subtraction
*	<code>float ← float * float</code>	Multiplication
/	<code>float ← float / float</code>	Division

Comparison operators

Operator	Expression type	Meaning
<code>!=</code>	<code>bool ← float != float</code>	Not equal
<code>==</code>	<code>bool ← float == float</code>	Equal
<code>></code>	<code>bool ← float > float</code>	Greater than
<code><</code>	<code>bool ← float < float</code>	Lower than
<code>>=</code>	<code>bool ← float >= float</code>	Greater or equal
<code><=</code>	<code>bool ← float <= float</code>	Lower or equal

1.3.2 float getters

getter	Type	Meaning
string	string	Returns a string representation of the float expression. [4.2 string] returns string "4.2".
cos	float	Returns the cosine of a float expression expressed in radian.
sin	float	Returns the sine of a float expression expressed in radian.
tan	float	Returns the tangent of a float expression expressed in radian.
cosDegree	float	Returns the cosine of a float expression expressed in degree.
sinDegree	float	Returns the sine of a float expression expressed in degree.
tanDegree	float	Returns the tangent of a float expression expressed in degree.
exp	float	Returns the exponentiation of a float expression.
logn	float	Returns the natural logarithm of a float expression.
log2	float	Returns the logarithm base 2 of a float expression.
log10	float	Returns the logarithm base 10 of a float expression.
sqrt	float	Returns the square root of a float expression.
power	float	This getter takes one argument, p. It returns the expression raised to the power of p.

1.4 The string data type

The `string` data type supports unicode. A literal string is delimited by a pair of ". Literal strings support special characters:

Escape sequence	Corresponding character
<code>\f</code>	form feed
<code>\n</code>	new line
<code>\r</code>	return
<code>\t</code>	horizontal tab
<code>\v</code>	vertical tab
<code>\\</code>	backslash
<code>\0</code>	null character
<code>\unnnn</code>	unicode character with code <i>nnnn</i> in hexadecimal
<code>\Unnnnnnnn</code>	unicode character with code <i>nnnnnnnn</i> in hexadecimal

1.4.1 string operators

The `string` data type supports the following operators:

Binary operator

Operator	Expression type	Meaning
<code>+</code>	<code>string ← string + string</code>	Concatenation

Comparison operators

Operator	Expression type	Meaning
!=	<code>bool ← string != string</code>	Not equal
==	<code>bool ← string == string</code>	Equal
>	<code>bool ← string > string</code>	Greater than
<	<code>bool ← string < string</code>	Lower than
>=	<code>bool ← string >= string</code>	Greater or equal
<=	<code>bool ← string <= string</code>	Lower or equal

1.4.2 string getters

getter	Type	Meaning
HTMLRepresentation	string	Returns a representation of the string suitable for an HTML encoded representation. ‘&’ is encoded by <code>&amp;</code> ; , ‘”’ by <code>&quot;</code> ; , ‘<’ by <code>&lt;</code> ; and ‘>’ by <code>&gt;</code> ; .
identifierRepresentation	string	Returns an unique representation of the string conforming to a C identifier. Any Unicode character that is not a latin letter is transformed into its hexadecimal code point value, enclosed by ‘_’ characters. This representation is unique: two different strings are transformed into different C identifiers. For example: <code>value3</code> is transformed to <code>value_33_</code> ; <code>+=</code> is transformed to <code>_2B__3D_</code> ; <code>An_Identifier</code> is transformed to <code>An_5F_Identifier</code> .
fileExists	bool	Returns <code>true</code> if a file exists at the target path, <code>false</code> otherwise.
length	integer	Returns the number of characters in the string
lowercaseString	string	Returns the lowercased representation of the string.
capitalized	string	if the string is empty, this getter returns the empty string; otherwise, it returns the string with the first character being replaced with the corresponding upper case character.
uppercaseString	string	Returns uppercased representation of the receiver

getter	Type	Meaning
leftSubString	string	Returns the sub-string from the beginning of the target and with the number of characters passed as argument. If the sub-string is longer than the target, the target is returned. ["Hello World !" leftSubString : 5] returns "Hello".
rightSubString	string	Returns the sub-string from the end of the target and with the number of characters passed as argument. If the sub-string is longer than the target, the target is returned. ["Hello World !" leftSubString : 11] rightSubString: 5] returns "World".
subString	string	Returns the sub-string from the index passed as first argument and with the number of characters passed as second argument. If the index is out of the target, the empty string is returned. If the number of characters is greater than the sub-string, the sub-string is returned. ["Hello World !" subString : 6, 5] returns "World". ["Hello" subString : 10, 3] returns the empty string. ["Hello" subString : 2, 10] returns "llo".
reversedString	string	Returns a mirrored string. ["Hello World !" reversedString] returns "!dlroW olleH".
componentsSeparatedByString	list	This getter takes one string argument: separator . The target is cut into pieces according to the separator and a list of the pieces is returned. ["Hello World !" componentsSeparatedByString : " "] returns @("Hello", "World", "!").
columnPrefixedBy	string	This getter takes one string argument: prefix . Return the target with each line prefixed by prefix. ["Hello\nWorld" columnPrefixedBy : "#_"] returns "#_Hello\n#_World".

getter	Type	Meaning
wrap	string	Wraps the target to a width. This getter takes two int arguments: width and shift . The target is assumed to contain paragraphs separated by <code>\n</code> . Returns the target with each paragraph wrapped to width . In addition, each line of the paragraph except the first one is prefixed by shift spaces. <code>["Hello beautiful World.\nHow are you" wrap : 6, 2]</code> returns <code>"Hello\n__beautiful\n__World\nHow\n__are\nyou"</code> .
subStringExists	bool	This getter takes one argument, subString . It returns true if the sub-string subString is found in the target, false otherwise.
replaceString	string	This getter takes two argument, find and replace . It returns the target where each occurrence of find is replaced by replace .

1.5 The `bool` data type

A true literal bool can be written as **true** or **yes** and a false literal bool can be written as **false** or **no**.

1.5.1 `bool` operators

The `bool` data type supports the following operators:

Unary operator

Operator	Expression type	Meaning
<code>~</code>	<code>bool ← bool</code>	logical not

Binary operator

Operator	Expression type	Meaning
<code>&</code>	<code>bool ← bool & bool</code>	logical and
<code> </code>	<code>bool ← bool bool</code>	logical or
<code>^</code>	<code>bool ← bool ^ bool</code>	logical exclusive or

Comparison operators

For comparison operators, `false` is considered to be lower than `true`.

Operator	Expression type	Meaning
<code>!=</code>	<code>bool ← bool != bool</code>	Not equal
<code>==</code>	<code>bool ← bool == bool</code>	Equal
<code>></code>	<code>bool ← bool > bool</code>	Greater than
<code><</code>	<code>bool ← bool < bool</code>	Lower than
<code>>=</code>	<code>bool ← bool >= bool</code>	Greater or equal
<code><=</code>	<code>bool ← bool <= bool</code>	Lower or equal

1.5.2 bool getters

getter	Type	Meaning
<code>trueOrFalse</code>	string	Returns a string representation, " <code>true</code> " or " <code>false</code> " of the bool expression.
<code>string</code>	string	Returns a string representation, " <code>true</code> " or " <code>false</code> " of the bool expression.
<code>yesOrNo</code>	string	Returns a string representation, " <code>yes</code> " or " <code>no</code> " of the bool expression.
<code>TRUEOrFALSE</code>	string	Returns a string representation, " <code>TRUE</code> " or " <code>FALSE</code> " of the bool expression.
<code>YESOrNO</code>	string	Returns a string representation, " <code>YES</code> " or " <code>NO</code> " of the bool expression.
<code>int</code>	int	Returns an int representation, 1 or 0 of the bool expression.