Casual Language Documentation
Compiling Techniques 2019–2020

João David
49448

26/06/2020

1 Getting Started

1.1 Syntax Highlighter

The Casual Language has a syntax highlighter extension for Visual Studio Code. This tool also has auto-complete functionality, that can be used by pressing CTRL+SPACE after writing some characters, if there are any words previously written using those characters, they will appear (e.g. variable and function names). The extension can be installed through the "casual-syntax-highlighter-0.0.1.vsix" file.

1.2 Running the compiler

The casualc script accepts as argument the casual source file that will be compiled and then creates the binary file with the same name of the source file (without the .cas).

Since this language supports imports, all the source files involved in the process must be passed as argument, so that the compiler may link them and output a single binary file, that will have the name of the first source code passed as argument.

For instance, "casualc hello.cas math.cas util.cas" will output the binary hello. From all the source files passed to the compiler, there must be exactly one defined method with the name "main". This will be the method executed first by the binary file.

1.3 Writing Casual code

The Casual language has 12 built-in functions that can be used by the programmer, in order to use them, they have to be declared. The following list summarizes all existent declarable functions, they allow the user to print, create arrays, and matrixes for each data type (Int, Float, Bool and String).

- decl printInt(i:Int):Void
- decl printFloat(f:Float):Void
- decl printBool(b:Bool):Void
- decl printString(s:String):Void
- decl new_int_array(size:Int):[Int]
- decl new_float_array(size:Int):[Float]
- decl new_bool_array(size:Int):[Bool]
- decl new_string_array(size:Int):[String]
- decl new_int_matrix(size:Int, size2:Int):[[Int]]
- decl new_float_matrix(size:Int, size2:Int):[[Float]]
- decl new_bool_matrix(size:Int, size2:Int):[[Bool]]
- decl new_string_matrix(size:Int, size2:Int):[[String]]

2 Implementation Decisions

2.1 Operator Precedence

Precedence order. When computing the expression $3 + 3 * 5$, the operator with higher precedence goes first, in this case its the multiplication, the expression is evaluated as if it was $3 + (3 * 5)$.

Associativity. When an expression has two operators with the same precedence, it is evaluated according to its associativity. For instance, the expression $72/2/3$ is computed as if it was $(72/2)/3$, because the division operator has left to right associativity. Other operators are not associative, therefore, they can't share the same operand with other operators in the same level of precedence. The expression $3 < 4 >= 4$ is invalid.

The following table summarizes all the information regarding precedence and associativity in Casual, a higher level means higher precedence.

Level	Operator	Description	Associativity
9	[ ] ( )	access array element parentheses	left to right
8	$-$!	unary minus unary logical NOT	right to left
7	$*$ / $\%$	multiplicative	left to right
6	$+$ $-$	additive	left to right
5	$<$ $<=$ $>$ $>=$	relational	not associative
4	$==$ $!=$	equality	left to right
3	$\&\&$	logical AND	left to right
2	$\|\|$	logical OR	left to right
1	$=$	assignment	right to left

2.2 Linking Source files