MSDscript Users Guide

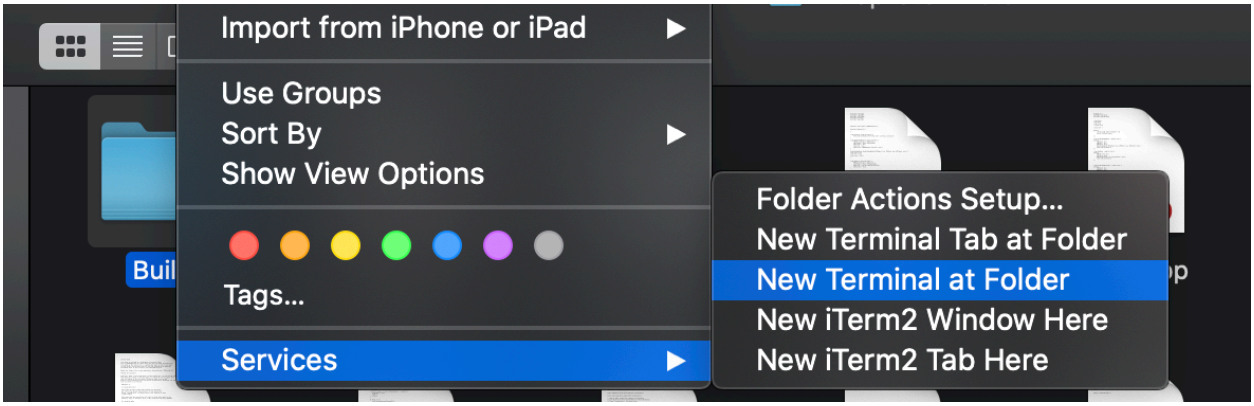
MSDScript is a scripting language written in C++. It is used to interpret user input of mathematic operations and returns either an integer value, or an optimized version of the input received. This package contains two parts, a command line implementation of the language and a library that can be inserted into existing projects. MSDscript can be used as a way to provide users a basic language for user input.

It includes support for addition, multiplication, booleans, functions and recursion.

Installation

1. Download CMake from https://cmake.org/download/
2. Install CMake
3. Unzip the "MSDScript.zip" file.
4. Open terminal in the unzipped directory by right clicking on the build directory and choosing the Services tab, then "New Terminal at folder.
5. Type "cmake .." And press enter.
6. Type "cmake --build ."
7. MSDscript executable has now been created in the build folder and can be run by double clicking msdscript, or by navigating to the folder in the terminal and typing, "./msdscript". libmsdscriptl library has also been created and is ready to be embedded into your project.
8. If you are linking the library into a c++ project, add the file name in the header and make sure that the libmsdscriptl.a is saved at the correct path, or add that path to your project where it is saved so your IDE can find the file. For more information on this step, use the guide at the following link.

   https://automaticaddison.com/how-to-add-an-external-c-library-to-your-project/
9. On page 9 a guide for using the library in your own projects is included.



(figure for step 4.)

Getting started with msdscript

There are three parts to this system.

- Interpreter, which evaluates an MSDscript expression (defined below) and returns a value. The default operation mode.
- Optimizer, which simplifies an MSDscript expression and returns a simpilified MSDscript expression. Enabled with the —opt tag “./msdscript --opt”
- Step, same as interpreter but takes much less space on the stack, use this if you are getting a segfault. Enabled with the —step tag “./msdscript --opt”

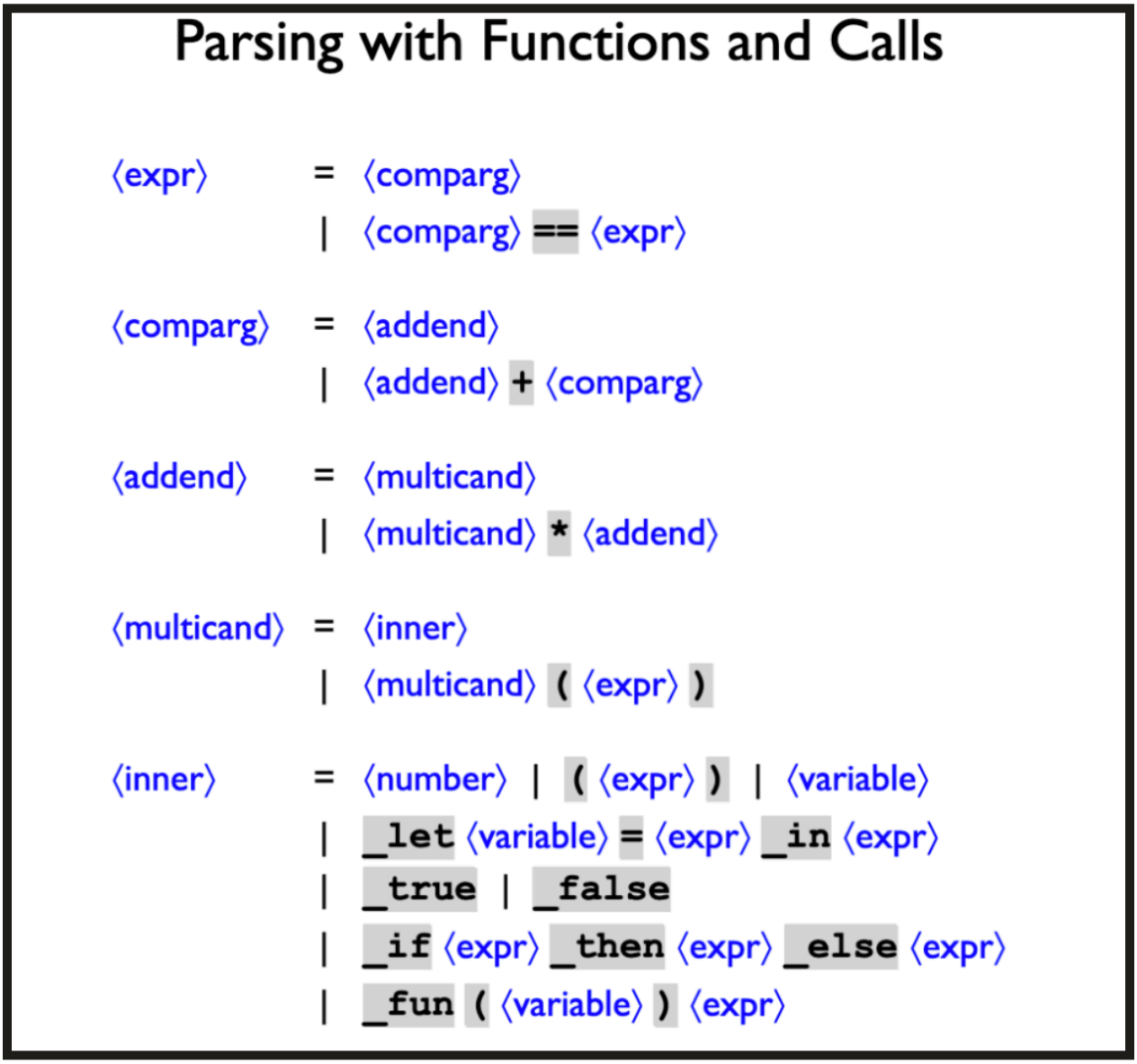
Entering an Expression

After setting what mode you want, an expression can be entered for evaluation. Expressions follow the grammar and there are examples below to show the format that the program.

Expression Definition and General Input Guidelines

Some general input guidelines.

- All valid input follows the grammar, the grammar and examples are a good reference for creating good input.
- Some input is legal but not useful and will return a phrase indicating that something that does not make sense has been input. For example, multiplying booleans _true * _true will not evaluate and will return the message “no multiplying booleans”. Keep this in mind if you receive this or similar messages.



Input Examples and Program Behavior

To get started, run the program in the command line using the steps above. After executing the program you will have an open prompt. To interpret or optimize an expression, type the expression you want hit enter and then hit ctrl + d. The program will then interpret or optimize the given expression.

*Note: If the input expression evaluates to a single character value, it will be returned with a D on the end of it. The terminal causes this and msdscript has not yet been updated to account for it so for now the D can be ignored. An example of this, if you run 1 + 2 in the interpret mode, the value 3D will be the output. All of the example outputs in this document ignore the D.

Expressions can include integers and the “+” and “*” operator, parentheses are also valid input. Division and subtraction our not supported, but subtraction can be accomplished by adding negative integers. When evaluating input expressions without variables, all modes will return the same values.

Program Mode	Example Input Expression	Output
Interpreter/Step	1+2	3
	1+-2	-1
	3*2+2	8
	3*(2+2)	12
	4*(2+4)*6*(6+7)	1872
Optimize	1+2	3
	1+-2	-1
	3*2+2	8
	3*(2+2)	12
	4*(2+4)*6*(6+7)	1872

Along with integers, Variables can be entered into expressions. They have to be put in using the format _let x = (integer) _in (integer) + 5. _let and _in are keywords and must be input exactly, case matters, also `_let` binds for use after `_in`. Variable names can be any ascii values, lower or upper case. You can also nest variables, so that you can have multiple such as _let x = 1 _in _let y=2 _in x + 2 will evaluate to 3.

One thing to note is that interp mode returns only values. Free variables are considered an error while in interp mode. x + 2 will evaluate to free variable: x. Optimize will evaluate to x + 2. A note on variable input, Variables only contain ASCII alphabetic characters (no numbers, special characters, or spaces). Variable names are case-sensitive.

Multiplication must be written as Val1 * Val2, not (Val1)(Val2) or Val1Val2.

Program Mode	Example Input Expressions (including variables)	Output
Interpreter/Step	_let x=4 _in x*4	16
	_let x = -6 _in x * x	36
	_let x=4 _in _let y=2 _in x+y	6
	x + 2 + 5	free variable: x
	_let x=4 _in _let y=2 _in x+y+z	free variable: z
Optimize	_let x = -6 _in x * x	36
	1+-2	-1
	_let x=4 _in _let y=2 _in x+y	6
	x + 2 + 5	x + 7
	_let x=4 _in _let y=2 _in x+y+z	4 + 2 + z

MSDscript also supports booleans, '_true' and '_false'. Some expressions produce booleans, such as 1==1 will return the _true boolean. Conditionals are setup using the form _if _true _then x _else y. This can be used to create more complex expressions.

Functions are also supported in MSDscript. They can be added using the form "_let f = _fun (x) x + 1 _in f(10)" x is the function's name, x+1 is the function's body and in f(10), 10 is the variable that is input. Functions do not have to use _let, they are considered values and a "_fun" expressions can appear anywhere, including as a function argument, function result, or directly in a function-call's to-be-called position. The additional examples on the sheet 6 show some good examples of more complex functions and how they can be assembled.

Optimizing a function will not interpret it. Optimization produces a semantically equivalent expression that is no larger than the input expression, so for functions it will simplify if it can, but if it can't it will return the given function. It may add parentheses that 2 not in the input function.

Program Mode	Example Input Expressions (including variables)	Output
Interpreter/Step	1 == 1	_true
	1 == 2	_false
	_if _true _then 5 _else 6	5
	_let f = _fun (x) x + 1 _in f(10)	11
	_let f = _fun (x) x * x _in f(5)	25
Optimize	1 == 1	_true
	1 == 2	_false
	_if _true _then 5 _else 6	5
	_let f = _fun (x) x + 1 _in f(10)	_let f = (_fun (x) x + 1) _in f(10)
	_let f = _fun (x) x * x _in f(5)	_let f = (_fun (x) x * x) _in f(5)

This includes all of the operations msdscript currently supports. Below are some examples of how these can be combined into more interesting input.

Program Mode	Example Input Expressions (recursion example, factorial, and fibbonacci) (Step example)	Output
Interpreter/ Step (factorial)	_let factrl = _fun(factrcl) _fun (x) _if x == 1 _then 1 _else x * factrl(factrl)(x + -1) _in _let factorial = factrl(factrl) _in factorial(5)	120
Interpreter/ Step (Fibonacci)	_let fib = _fun (fib) _fun (x) _if x == 0 _then 1 _else _if x == 2 + -1 _then 1 _else fib(fib)(x + -1) + fib(fib)(x + -2) _in fib(fib)(10)	89
Interpreter	_let countdown = _fun(countdown) _fun(n) _if n == 0 _then 0 _else countdown(countdown)(n + -1) _in countdown(countdown)(1000000)	Stack Overflow
Step		0

As alluded to in the beginning of this document there are a number of warning messages that are possible to receive when you input something that is parsed correctly but doesn't equal something that can be interpreted. There are also errors that come from bad input that cause the parser to fail. If you get one of these warnings consult the table below.

Error message	Cause
Integer addition between integers only.	Attempted to add a number with something that cannot be added, example. example, Bool, Function.
Integer multiplication between integers only.	Attempted to multiply a number with something that cannot be added, example. example, Bool, Function.
Function is not true or false, Not a bool.	Asked if function is true. Doesn't make sense.
Number is not true or false, Not a bool.	Asked if number is true. Doesn't make sense.
Value, no function here.	Tried to call a value like a function.
Bool, No function here.	Tried to call a bool like a function.
No adding booleans.	Tried to add bools. Cannot add bools.
No multiplying booleans.	Tried to multiply bools. Cannot multiply bools.
No adding function Vals.	Tried to add functions. Cannot add functions.
No multiplying function Vals.	Tried to multiply functions. Cannot multiply functions.
free variable: (variable name)	Free variables are not allowed in interpret mode. Must have a value assigned.
Step can't continue, exiting.	Step failed, check inputs and ending condition of input if loop.
Done count, wrong location	Step failed, check inputs and ending condition of input if loop.
(was expected after _fun	Missing parentheses.
) was not found in function.	Missing parentheses.
expected a digit or open parenthesis at	Inner statement not formatted correctly.

Library Guide

This guide is provided as a supplement to the User Guide. It is recommended that all users start with the user guide then move onto this guide as the library file is created when cmake is run. If you have run the install instructions you will see that along with the msdscript executable there is also a library file name MSDscriptlib.

Embed this library file in your application Using #includeÂ "MSDscriptlib".

Using the library

msdscript, takes a string and parses it into expressions, you can see the grammar that msdscript follows in the user guide. It follows all of the input rules that are covered in the user guide. The parser takes a string and parses it into expressions that can then be interpreted or optimized. The way that the command line tool works is as follows.

```
PTR(Expr) e;

e = parse(std::cin);
```

Parse takes an input stream that contains an expression, and returns the parsed representation of that expression. ThrowsÂ runtime_errorÂ for parse errors. This function is used to parse input.

After input has been parsed you will need to choose which mode to run; optimize, step, or interpret. This is done using the following commands.

For optimize mode:

```
std::cout << e->optimize()->toString();
```

For interpret mode:

```
std::cout << e->interp(Env::empty)->to_string();
```

For step mode:

```
std::cout << Step::interp_by_steps(e)->to_string();
```

See the documentation for more in depth descriptions of the different expression types and more information about how they can be used.