WHIRL PROGRAMMING LANGUAGE

Whirl was designed with the following state-of-the-art features in mind:

- Inheritance
- Simplicity
- Ease of use
- XML
- Maintainability
- Polymorphism
- Flexibility
- Power
- Subtraction
- Grilled Chicken

Well if you got enamoured by the above features, you did not get the sarcasm!

Whirl in essence has only two instructions 0 and 1.So when you check out a Whirl program; it may seem just an incomprehensible stream of 0s and 1s.

For instance have a look at this:

```
011001000111100010011111000001111000011111000001100011100000100110011000111100
```

Now what the beautiful code above does is that it takes two inputs from the user and adds them and prints the result!

Obviously you might be wondering how does it do operations or differentiate between specific commands among the myriad of 0s and 1s.To do this the designer of whirl has provided two rings- Operations ring and Maths ring with 12 commands each and the 1s are used to rotate the rings and 0s are used to change the direction of rotation or execute the current command on the current ring.

So basically when you write a program in whirl you are just playing with the two rings available to you by executing commands on the rings, switching between rings and so on. The rings are in fact your only point of contact with the machine and what's going on with your program.

Now let's delve in the dizzy world of whirl. But before that- a caution by the inventor of whirl- Mr. Sean Heber himself---

WARNING: Programming Whirl may be hazardous to your sanity!

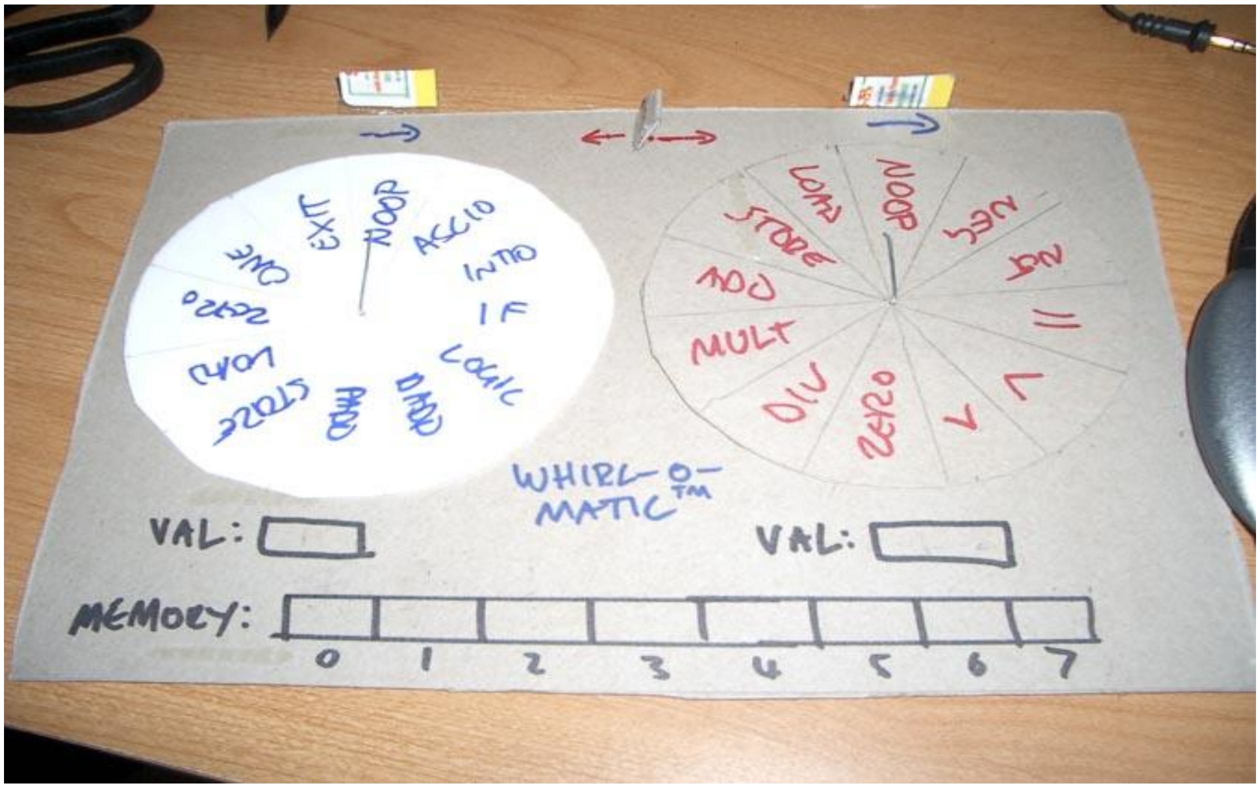
So fasten your seatbelts and let's move on….

The Language

Whirl: Whirl as it is very clear is an esoteric language and not meant for normal application programming unless you have loads of free time on your hands with nothing to occupy your programming brain. To “help” the programmers the whirl environment has generously given the following tools:

- An "infinite" supply of data memory
- An ability to change the program's instruction position
- One ring of 12 Control/Logic/IO operations
- One data storage slot for the operations ring
- One ring of 12 Math functions
- One data storage slot for the math ring
- Two easy-to-remember instructions

An important point to notice is that apart from the 12 functions that each of the two rings provides, they also have a data slot (henceforth called “value”) each. The value of the math ring is the basis on which all the mathematical operations are performed with the memory value (henceforth called “memval”). Similarly the value of the operations ring is used for logical operations.



A Graphical representation of the whirl machine

When a Whirl program begins, the operations ring is set as active and it is in the “Noop” position. The memory is initialized to 0. The operations ring's data storage slot is set to 0. The math ring is at the Noop position. The math ring's data storage slot is set to 0. The direction setting of both the operations ring and the math ring are set to clockwise.

WHIRL QUICK START GUIDE

Whirl Instructions	
1	Rotate the current ring in the current direction.
0	Reverse the direction of rotation of the active ring. If previous instruction was a 0 *and it did not trigger an execution*, then the currently selected command on the currently active ring is executed and the active ring is switched to the other ring. To get two executions in a row (one on the current command of each ring) requires 0000 and not 000.

Note... In the following documentation, **value** refers to the ring's data value and **memval** refers to the value of the currently selected block of memory.
All commands are listed in clockwise order starting with the default command (the command the ring is set to when a program begins).

Active ring: The ring which is used currently. Only the active ring can execute commands. As said 0000 executes the current command of the active ring and switches the active ring.

Default direction of movement of rings: For both the rings the default direction of movement is clockwise which can be changed with a single 0 instruction.

Here it has to be noted that whenever you switch between the active rings, then the direction of movement and the current command of the previous ring is retained.

After understating how to rotate a ring and execute commands on it as well as switch between the active rings let us now look at the various commands that each of the rings provides.

OPERATIONS RING COMMANDS

Operations Ring Commands	
Noop	Has no effect.
Exit	Terminates the program.
One	Sets **value** to 1.
Zero	Sets **value** to 0.
Load	Sets **value(location)** to **memval**.
Store	Sets **memval(location)** to **value**.
PAdd	Adds **value** to the current program position pointer (a jump).
DAdd	Adds **value** to the current memory position pointer (change memory location).
Logic	If **memval** is 0, then **value** is set to 0. Otherwise, **value** is set to **value** AND 1. This is a logical AND.
If	If **memval** is not 0, then add **value** to program position pointer (see PAdd).
IntIO	If **value** is 0, set **memval** to integer number read from stdin. Otherwise print **memval** to stdout as an integer.
AscIO	If **value** is 0, set **memval** to ASCII character read from stdin. Otherwise print **memval** to stdout as an ASCII character.

MATH RING COMMANDS

Math Ring Commands	
Noop	Has no effect.
Load	Sets **value(location)** to **memval**.
Store	Sets **memval(location)** to **value**.
Add	Sets **value** to **value** plus **memval**.
Mult	Sets **value** to **value** multiplied by **memval**.
Div	Sets **value** to **value** divided by **memval**.
Zero	Sets **value** to 0
<	If **value** is less than **memval**, then sets **value** to 1. Otherwise sets **value** to 0.
>	If **value** is greater than **memval**, then sets **value** to 1. Otherwise sets **value** to 0.
=	If **value** is equal to **memval**, then sets **value** to 1. Otherwise sets **value** to 0.
Not	If **value** is not 0, then sets **value** to 0. Otherwise sets **value** to 1.
Neg	Sets **value** to **value** multiplied by -1 (inverse).

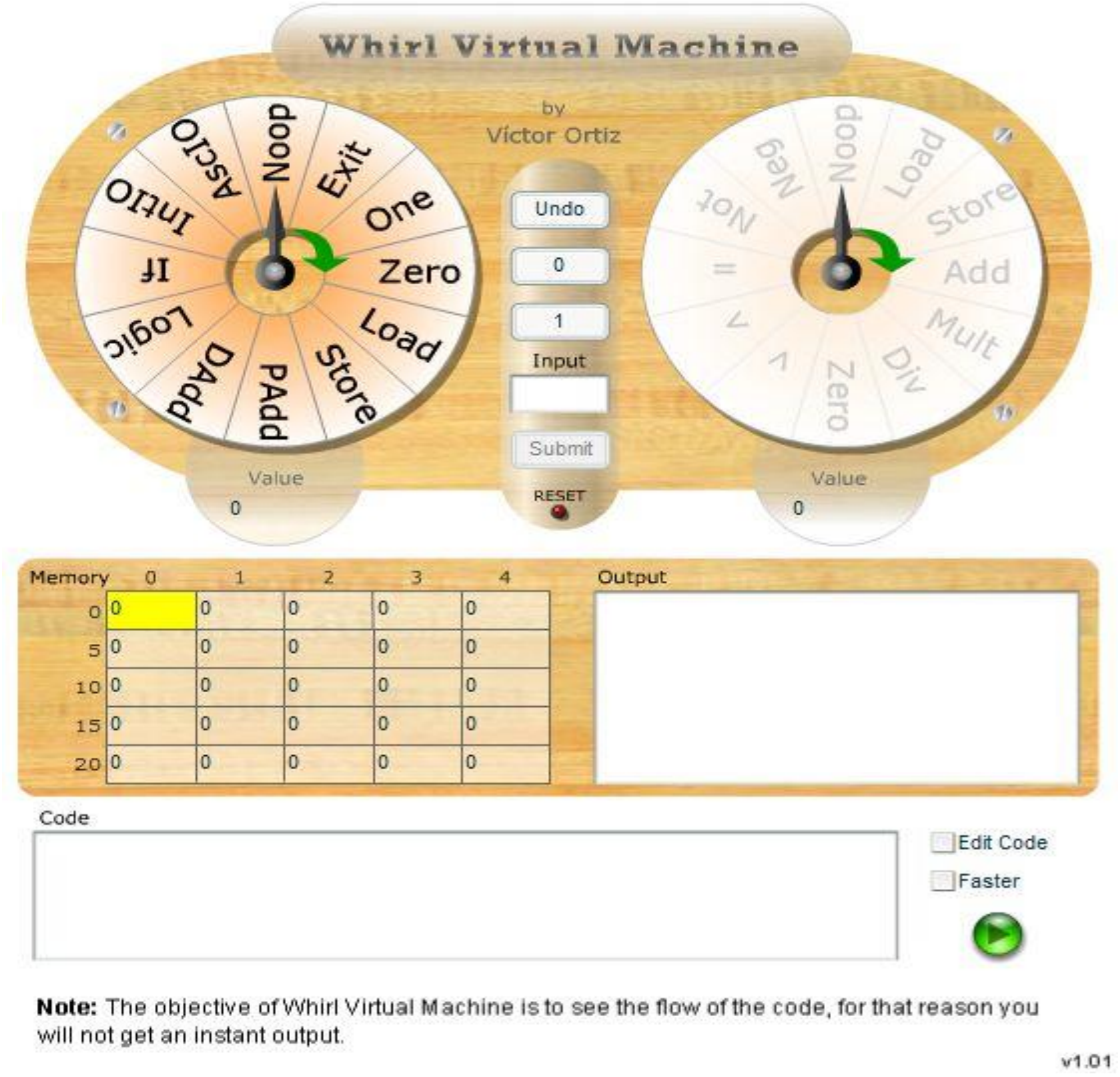
A Whirl program file consists of a series of 0 and 1s. Anything that isn't a 0 or a 1 is ignored. So commenting is pretty easy -- just beware of numbers that might have a 1 or 0 in them!

<u>Some examples</u>:-

One of the main trouble that the whirl programmers have to face is remembering the position as well as the direction of last active ring at each step. Once you make a mistake in that regard somewhere, then troubleshooting the code becomes as big a headache as you can possibly imagine.

In order to take our minds off such “mundane” things, the site: http://www.bigzaphod.org/whirl/ has come out with a graphical, flash based whirl virtual machine. This machine shows the dynamic execution of code and is a great tool for learning.

Here’s a snapshot:



Memory	0	1	2	3	4
0	0	0	0	0	0
5	0	0	0	0	0
10	0	0	0	0	0
15	0	0	0	0	0
20	0	0	0	0	0

Note: The objective of Whirl Virtual Machine is to see the flow of the code, for that reason you will not get an instant output.

Examples with explanation--- (it is suggested you use the flash based machine in all programming for ease)

1) Do 1+1 and print the results:

Solution:

00 – run ops.noop, switch to math ring
0 - math::ccw
11 -rotate to math.not
00 - run math.not, switch to ops ring
00 - run ops.noop, switch to math ring
0 - math::cw
1111 - rotate to math.store
00 - run math.store, switch to ops ring
00 - run ops.noop, switch to math ring
1 - rotate to math.add
00 - run math.add, switch to ops ring
00 - run ops.noop, switch to math ring
0 - math::ccw
1 - rotate to math.store
00 - run math.store, switch to ops ring
11 - rotate to ops.one
00 - run ops.one, switch to math ring
00 - run math.store, switch to ops ring
0 - ops::ccw
1111 - rotate to ops.IntIO
00 - run ops.IntIO, switch to math ring

2) The program at the beginning of this tutorial ,i.e., do a+b and print the result(a & b entered by user)

01100-ops: ccw, takes the “ring command pointer” to ops.Intio and runs it and stores input in 1st memory block .Ring switched to math

1000-goes to maths.load and sores the input in 1st memory block(memval) to value of maths ring. Ring switched and direction ops:cw

111100- goes to ops.one, runs it, switch ring to math

0100-math:ccw,goes to noop runs math.noop, switch to ops

1111100-goes to ops.dadd , runs it so memory block shifts 1(value),switch to math ring

000-runs math.noop, switch to ops, ops:ccw

111100- goes to ops.zero, runs it, switch to math ring

00-runs math.noop, switch to ops

1111100-goes to ops.intio,takes input and stores to current memory block(2nd), switch to math

00-run math.noop,switch to ops

01100-ops:cw, go to ops.noop, run it , switch to math

01110000-math:cw, go to math.add, run it, switch to ops, run ops.noop, switch back to math

0100-math:ccw, go to math.store, run it, swith to ops

1100-go to ops.one, run it, switch to math

1100-go to math.noop, run it, switch to ops

0111100-ops:ccw, go to ops.Intio, run it

Some more examples….

3) Code for a*b

0110010001111000100111110000011110000

11111000001100011110000011001100110001 11100

4) Code for a /b

011001000111100

0100111110000011110 0

0011111000001100011111 0000

01110011001100011110 0

5) Code to find 1+2+3+…+n (n entered by user)

```
011001000110011100000100001001100110011111000001100010011111001100000100001
110011111000111001110001100011100010001111100010001111100011111001111100011
10001100
```