Features:

The asm reads from stdin and writes to stdout. It uses a postfix notation for arithmetic operations. Asm is a stack-based program that operates using the set of instructions defined below.

Operation:

The contents of a file will be redirected to the stdin of asm. Asm will operate on the contents of this file, based upon the usage of the instruction set, documented below. It will continue to process stdin until the end of the file is reachd, at which point execution will terminate. Asm can also be used interactively. To do this, simply use the instruction set documented below. To signal the end of the set of program instructions, use the instruction stop. After that, use the keyword end to signal the end of interactive input. At this point, the instructions will be executed, and any output generated by them will printed to the screen.

Instruction Set:

ld: Load

Push the data that is stored in the memory location represented by the most recent Pop() onto the stack.

Example:

```
awoods:~$ ./asm
push 5
push x
ld
wr
stop
end
5
```

st: Store

Takes the value given by the next Pop(), and uses it as a memory location. It then stores the value given by the following Pop() into the memory location referenced by the first Pop().

```
awoods:~$ ./asm
push x
push 10
st
push x
ld
wr
stop
end
10
```

push: Push

Pushes the value given as a parameter onto the program stack.

pop: Pop

Removes the top element of the program stack.

dup: Duplicate

Duplicates the element on the top of the stack by popping it off, and then pushing it back onto the stack twice.

```
awoods:~$ ./asm
push 10
dup
wr
wr
stop
end
10
10
```

swap: Swap

Pops two elements off of the stack. After both pops have completed, the element that was popped off the stack first is pushed back onto it, followed by the second element to be popped off the stack.

A, B -> B, A

```
awoods:~$ ./asm
push 5
push 10
wr
wr
stop
end
5
10
```

add: Add

Pops two elements off of the stack, adds them together, and pushes the result back onto the stack.

Example:

```
awoods:~$ ./asm
push 5
push 6
add
```

```
wr
stop
end
11
```

sub: Subtract

Because this program uses postfix notation, the top two elements on the stack must first be switched. First Iswap() is called, reversing the order of the top two elements. Next two elements are popped off of the stack. The second element is subtracted from the first element. The result of the subtraction is then pushed back onto the stack.

Postfix notation: a b -
Stack order: - b a|
After swap: a b|

Once the swap has occurred, the subtraction takes the form of

a - b

After this computation has been performed, the result of the subtraction is pushed back onto the top of the stack.

Example:

```
awoods:~$ ./asm
push 7
push 2
sub
wr
stop
end
5
```

mul: Multiply

Since the order of multiplication does not matter, two pops are performed, the values from each pop are multiplied together, and the product is stored back on the stack.

Example:

```
awoods:~$ ./asm
push 5
push 4
mul
wr
stop
end
20
```

div: Divide

Because this program uses postfix notation, the top two elements on the stack must first be switched. First Iswap() is called, reversing the order of the top two elements. Next two elements are popped off of the stack. The first element becomes the numerator of the division, and the second element the denominator. The result of the division is then pushed back onto the stack.

Postfix notation: a b /
Stack order: / b a|
After swap: a b|

Once the swap has occurred, the division takes the form of

a / b

After this computation has been performed, the result of the division is pushed back onto the top of the stack.

Example:

```
awoods:~$ ./asm
push 100
push 2
div
wr
stop
end
50
```

mod: Modulus

Because this program uses postfix notation, the top two elements on the stack must first be switched. First Iswap() is called, reversing the order of the top two elements. Next two elements are popped off of the stack. The first element is used as the numerator in the modulus operation, and the second element is used as the denominator. The result of the modulus is then pushed back onto the stack.

Postfix notation: a b %
Stack order: % b a|
After swap: a b|

Once the swap has occurred, the modulus operation takes the form of

a % b

After this computation has been performed, the result of the modulus operation is pushed back onto the top of the stack.

Example:

```
awoods:~$ ./asm
push 5
push 2
mod
wr
stop
end
1
```

not: Not

The not operation is used to negate a value. A value is first popped off of the stack, then its value is negated. This negated value is then pushed back onto the stack.

Example:

```
awoods:~$ ./asm
push 1
not
wr
stop
end
0
```

jmp: Jump

Sets the value of the program counter in order to jump to a specific set of instructions. This is done by setting the program counter equal to the value of the next Pop() function.

jz: Jump Zero

Two Pop() commands are used for the jump zero command. The first contains the value that the program counter will be conditionally set to. The second is the value that the condition is dependent upon. If the value of the second Pop() is 0, then the program counter will be set equal to the value of the first Pop(), otherwise nothing happens.

jp: Jump Positive

Two Pop() commands are used for the jump positive command. The first contains the value that the program counter will be conditionally set to. The second is the value that the condition is dependent upon. If the value of the second Pop() is positive, then the program counter will be set equal to the value of the first Pop(), otherwise nothing happens.

jn: Jump Negative

Two Pop() commands are used for the jump negative command. The first contains the value that the program counter will be conditionally set to. The second is the value that the condition is dependent upon.

If the value of the second Pop() is negative, then the program counter will be set equal to the value of the first Pop(), otherwise nothing happens.

rd: Read

Reads from stdin, and pushes the content onto the program stack.

wr: Write

Writes to stdout the value contained at the top of the program stack.

stop: Stop

The stop command tells asm that the end of the program has been reached.