# Getchl

Because why *shouldn't* we overuse getch()?

Getchl is an Esoteric Shell designed for WalrusOS, the world's first EsOS. Getchl's main source of input is the stdin, using the `getch()` command to get info from the keyboard. This provides a different way of interacting with the computer, through single keystrokes instead of full-length commands terminated by returns.

## Data Model

In one of my favorite data model families, the model behind Getchl is a both-sides-infinite tape of byte stacks with a scalar accumulator.

## Commands

### Terminology

| Term | Meaning |
|---|---|
| Stack | LIFO queuing mechanism |
| Queue | FIFO queuing mechanism |
| Tape | A right-infinite array with an index |
| Pop | Take the value off the top of the stack and return it |
| Push | Put a value on the stack |
| Accumulator | A scalar that holds a value |
| 0gnirts | A null-terminated string on the stack (alternatively, if no null terminator is present, the entire stack as a string) (first character of string at top of stack for technical reasons) |
| Mode | Alternative interpreter modes |
| Default mode | The standard interpreter mode |
| String mode | The mode where commands are ord()ed and pushed on the stack |
| Block mode | The mode where commands are saved to a block instead of executed (used for evaluating groups of commands later) |
| Prime mode | The mode where commands are slightly different for control purposes |

| | |
|---|---|
| Safe mode | The mode where commands do not execute when the stack is too short, instead returning 0 |

## Notes

- All values are modulo 256 (or not, for a unicode implementation)

## Command List

All commands are subject to change. Feel free to suggest such changes yourself.

| Hex | Ch | Name | Meaning |
|---|---|---|---|
| 20 | ' ' | NOP | Do absolutely nothing |
| 21 | ! | NOT | Push the logical inversion of a popped value |
| 22 | " | STR | Enter string mode |
| 23 | # | DEFINE | Pop a value, map it to the previous command (or block) |
| 24 | $ | DROP | Pop a value and discard it |
| 25 | % | MOD | Pop a, pop b, push b%a |
| 26 | & | AND | Pop a, pop b, push b&a |
| 27 | ' | PRIME | Enter prime mode for the following character |
| 28 | ( | START BLOCK | Start forming a block |
| 29 | ) | END BLOCK | Terminate a block |
| 2A | * | MULT | Pop a, pop b, push b*a |
| 2B | + | ADD | Pop a, pop b, push b+a |
| 2C | , | INP | Get a single character as input and pop its ASCII value on the stack (fail if EOF) |
| 2D | - | SUB | Pop a, pop b, push b-a |
| 2E | . | PRINT | Print the character whose ascii value is on the ToS |
| 2F | / | DIV | Pop a, pop b, push b//a |
| 30 | 0 | ZERO | Push 0 |
| 31 | 1 | ONE | Push 1 |
| 32 | 2 | TWO | Push 2 |

| | | | |
|---|---|---|---|
| 33 | 3 | THREE | Push 3 |
| 34 | 4 | FOUR | Push 4 |
| 35 | 5 | FIVE | Push 5 |
| 36 | 6 | SIX | Push 6 |
| 37 | 7 | SEVEN | Push 7 |
| 38 | 8 | EIGHT | Push 8 |
| 39 | 9 | NINE | Push 9 |
| 3A | : | DUP | Pop a, push a, push a |
| 3B | ; | TERMINATE | End session |
| 3C | < | LT | Pop a, pop b, push 1 if b<a, else push 0 |
| 3D | = | EQ | Pop a, pop b, push 1 if b=a, else push 0 |
| 3E | > | GT | Pop a, pop b, push 1 if b>a, else push 0 |
| 3F | ? | IF | Pop a, pop b, pop c, if a!=0 push b, else push c. |
| 40 | @ | REDUCE | Repeat the previous instruction (or block) until it fails |
| 41 | A | A | Push 10 |
| 42 | B | B | Push 11 |
| 43 | C | C | Push 12 |
| 44 | D | D | Push 13 |
| 45 | E | E | Push 14 |
| 46 | F | F | Push 15 |
| 47 | G | GLOBAL | Create a variable (pop v, pop 0gnirts name, map name to v) |
| 48 | H | | |
| 49 | I | INVERT | Invert the stack<br><br>`st2 = Stack()`<br>`while len(stack) > 0:`<br>`  st2.push(st.pop())`<br>`st = st2` |

| 4A | J | TAPE RELATIVE JUMP | Pop n, go n-127 cells to the left on the tape |
|---|---|---|---|
| 4B | K | | |
| 4C | L | LEFT | Go left on the tape |
| 4D | M | MAXIMIZE | Pop a, set maximum stack size to a (pushes fail when len(stack) = stack.max) (extraneous elements dropped) (0 means no bound) |
| 4E | N | | |
| 4F | O | | |
| 50 | P | | |
| 51 | Q | QUEUE | Toggle queuing mode (all PUSHes become ENQUEUEs, all POPs become DEQUEUEs) |
| 52 | R | RIGHT | Go right on the tape |
| 53 | S | | |
| 54 | T | | |
| 55 | U | | |
| 56 | V | | |
| 57 | W | | |
| 58 | X | | |
| 59 | Y | | |
| 5A | Z | | |
| 5B | [ | ROT | Pop a, pop b, pop c, push a, push c, push b |
| 5C | \ | SWAP | Pop a, pop b, push a, push b |
| 5D | ] | ROTCC | Pop a, pop b, pop c, push b, push a, push c |
| 5E | ^ | ACCUMULATE | Pop a and put it in the accumulator |
| 5F | _ | DECCUMULATE | Push value in the accumulator |
| 60 | ` | APPLY/ EVAL | Pop a value and evaluate it as ASCII |

| | | | |
|---|---|---|---|
| 61 | a | | |
| 62 | b | | |
| 63 | c | | |
| 64 | d | DIR | Change the current working directory (pop a 0gnirts s, cd to s) |
| 65 | e | | |
| 66 | f | FIND | Pop tf, Repeatedly pop a until a==tf, push a |
| 67 | g | GET | Inverse of G: Pop a 0gnirts, push the global with that name |
| 68 | h | | |
| 69 | i | IDENTITY | Pop a value and push it (NOP normally, but equivalent to ROLL in queueing mode) |
| 6A | j | JUMP | Pop a 0gnirts, jump back to that location and reexecute to the current point |
| 6B | k | | |
| 6C | l | LABEL | Get a 0gnirts, label this location as a jumpback location |
| 6D | m | | |
| 6E | n | | |
| 6F | o | OPEN | Pop a, get 0gnirts s from the stack, open the file s and associate it with value a |
| 70 | p | | |
| 71 | q | | |
| 72 | r | READ | Pop a, read the content of the file associated with a |
| 73 | s | REPLACE | Pop three strings off the stack, replace s[1] with s[2] in s[0], push s[0] onto the stack |
| 74 | t | | |
| 75 | u | | |
| 76 | v | | |
| 77 | w | WRITE | Pop a, pop 0gnirts s, write s to the file associated with a |

| | | | |
|---|---|---|---|
| 78 | x | | |
| 79 | y | | |
| 7A | z | | |
| 7B | { | COMMENT | Stop executing commands (??? do I need this for a shell?) |
| 7C | \| | OR | Pop a, pop b, push b\|a |
| 7D | } | DECOMMENT | Resume executing commands |
| 7E | ~ | SAFE | Toggle safemode |

## Prime Mode Commands

Prime mode is an interpreter mode where commands are executed with alternative meanings. Unless otherwise specified, the interpreter returns to default mode after execution.

### Single-prime mode

| Character | Name | Meaning |
|---|---|---|
| '@ | CONDITIONAL REDUCE | Pop value, negate it, execute previous command (or block) until a certain type of failure, fail if it fails on another failure (complicated, I know) |
| '? | CONDITIONAL EXECUTE | Pop a, execute the previous instruction (or block) if a != 0 |
| '; | FAIL | Pop a value and fail with its negated value |
| '` | RUN | Pop a value and execute the function defined to it |
| '( | CONTINUED PRIME | Enter continued prime mode (terminates on \)). All commands are interpreted as prime (e.g. `'(abc)` is equivalent to `'a'b'c`). |
| '' | DOUBLE PRIME | Enter double-prime mode (currently does nothing). **Does not return to default mode** |
| 'G | GLOBAL STRING | Pop a 0gnirts name, pop a 0gnirts s, map name to string s |
| 'g | GET STRING | Pop a 0gnirts, push the 0gnirts it's mapped to |
| 'L | RELATIVE LEFT JUMP | Pop n, go n cells to the left on the tape |
| 'R | RELATIVE | Pop n, go n cells to the right on the tape |

| | | |
|---|---|---|
| | RIGHT JUMP | |
| '\| | XOR | Pop a, pop b, push b^a |

## Failure Codes

Some commands can return failure codes when executed under certain conditions. Failure codes do *not* terminate the script. All failure codes are negative.

| Value | Meaning |
|---|---|
| -1 | Not enough values on stack |
| -2 | Attempted to push onto full stack |
| -3 | Divide or modulo by 0 |
| -4 | No previous command (start of script or block) |
| -5 | No input character |
| -6 | Attempt to print null character |
| -7 | Invalid varname |
| -8 | Invalid filename |
| -9 | Invalid label |

## Useful Constructs and Mnemonics

| Construct | Meaning |
|---|---|
| **<op>@** | Where op is any operator (+, -, *, /, %, &, \|, '\|), reduce the stack with that operation (+ sums the stack, * multiplies it, etc) |
| ,@ | Get an input string, terminated by EOF |
| .@ | Print a 0gnirts |
| $@ | Clear the stack |
| `@ | Eval a 0gnirts |
| &! | NAND |
| \|! | NOR |

| | |
|---|---|
| '\|! | XNOR |
| **<n>**:@ | Fill the remainder of the stack with value n (exclude n for existing ToS) (Make sure the stack has a max, or it will hang forever) |
| **<n>**(**<coms>**1-0\/$)@ {I think this is right} | Execute coms n times (Push n on stack, then execute coms and push 1 and subtract 1 from n, push 1, swap, and divide it by n and drop. This means it fails when n is equal to 0, thus terminating the loop) (note that commands must not change the stack, or at least must bury added values under the existing control values.) |