Contents

1 Basic Test Results 2

2 README 3

3 Array.jack 4

4 Keyboard.jack 5

5 Makefile 7

6 Math.jack 8

7 Memory.jack 11

8 Output.jack 14

9 Screen.jack 19

10 String.jack 23

11 Sys.jack 26

1 Basic Test Results

```
********** TEST START **********

  preparing sub.tar
dos2unix: converting file /tmp/bodek.A6A030/nand2tet/Project12/ransha/presubmission/testdir/stud/sub.tar/README to Unix form
  checking sub.tar
testing ArrayTest
Test ArrayTest passed! Woohoo
testing MathTest
Test MathTest passed! Woohoo
testing MemoryTest
Test MemoryTest passed! Woohoo
Good luck on the manual tests :) See you on the 10/07!

********** TEST END **********
```

2 README

```
nivkeren,ransha
=============================================================================
Niv Keren, ID 201478351, niv.keren@mail.huji.ac.il
Ran Shaham, ID 203781000, ran.shaham1@mail.huji.ac.il
=============================================================================

                   Project 12- Operating System
                  ------------------------------


Submitted Files
---------------
README        - This file.

Run command
-----------


Remarks
-------
* executed as suggested in the book design
* "If I'm not back in five minutes, just wait longer."
    Ace Ventura: Pet Detective
```

3 Array.jack

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/12/Array.jack

/**
 * Represents an array. Can be used to hold any type of object.
 */
class Array {

    /** Constructs a new Array of the given size. */
    function Array new(int size) {
        if (~(size > 0)) {
            do Sys.error(2);
        }
        return Memory.alloc(size);
    }

    /** De-allocates the array and frees its space. */
    method void dispose() {
        do Memory.deAlloc(this);
        return;
    }
}
```

4 Keyboard.jack

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/12/Keyboard.jack

/**
 * A library for handling user input from the keyboard.
 */
class Keyboard {
    static int KEYBOARD;
    static int MAX_STRING;

    /** Initializes the keyboard. */
    function void init() {
        let KEYBOARD = 24576;
        let MAX_STRING = 80;
        return;
    }

    /**
     * Returns the ASCII code (as char) of the currently pressed key,
     * or 0 if no key is currently pressed.
     * Recognizes all ASCII characters, as well as the following extension
     * of action keys:
     * New line = 128 = String.newline()
     * Backspace = 129 = String.backspace()
     * Left Arrow = 130
     * Up Arrow = 131
     * Right Arrow = 132
     * Down Arrow = 133
     * Home = 134
     * End = 135
     * Page Up = 136
     * Page Down = 137
     * Insert = 138
     * Delete = 139
     * ESC = 140
     * F1 - F12 = 141 - 152
     */
    function char keyPressed() {
        var char key;
        let key = Memory.peek(KEYBOARD);
        return key;
    }

    /**
     * Reads the next character from the keyboard.
     * waits until a key is pressed and then released, then echoes
     * the key to the screen, and returns the value of the pressed key.
     */
    function char readChar() {
        var char firstKey, key;
        let firstKey = 0;
        while (firstKey = 0) {
            let firstKey = Keyboard.keyPressed();
        }
        let key = firstKey;
        while (key = firstKey) {
            let key = Keyboard.keyPressed();
        }
        if ((firstKey > 31) & (firstKey < 129)) {
            do Output.printChar(firstKey);
        }
        return firstKey;
    }

    /**
     * Prints the message on the screen, reads the next line
     * (until a newline character) from the keyboard, and returns its value.
     */
    function String readLine(String message) {
        var char c;
        var String s;

        do Output.printString(message);
        let s = String.new(MAX_STRING);
        let c = 0;
        while (~(c = String.newLine())) {
            let c = Keyboard.readChar();
            if (c = String.newLine()) {
                return s;
            }
            if (c = String.backSpace()) {
                if (~(s.length() = 0)) {
                    do s.eraseLastChar();
                    do Output.backSpace();
                }
            }
            else {
                do s.appendChar(c);
            }
        }
        return s;
    }

    /**
     * Prints the message on the screen, reads the next line
     * (until a newline character) from the keyboard, and returns its
     * integer value (until the first non numeric character).
     */
    function int readInt(String message) {
        var String s;
        let s = Keyboard.readLine(message);
        return s.intValue();
    }
}
```

5 Makefile

```makefile
# --- Empty Makefile ---
SHELL=bash

all:
	@echo "Done."

JACK_EXT=.jack
JACK_COMPILER=../../tools/JackCompiler.sh
TAR_FILES=README Makefile *$(JACK_EXT)
TAR_FLAGS=-cvf
TAR_NAME=project12.tar
TAR=tar

tar:
	$(TAR) $(TAR_FLAGS) $(TAR_NAME) $(TAR_FILES)

JACK_FILES=*$(JACK_EXT)
TEST_DIR="Test/"
LINK=ln
links:
	@echo Creating test files...
	@for f in $(JACK_FILES); do \
		f_clean=$$(basename $$f $(JACK_EXT)); \
		f_dir=$$f_clean$(TEST_DIR); \
		$(LINK) $$f $$f_dir$$f || break; \
		echo Created link: $$f_dir$$f; \
	done

compile:
	@echo Compiling test directories...
	@for d in *$(TEST_DIR); do \
		$(JACK_COMPILER) $$d; \
	done


clean:
	@echo Removing test files...
	@find . -mindepth 2 -maxdepth 2 -name "*$(JACK_EXT)" ! -name "Main.jack" -print -delete
	@echo Removing vm files...
	@find . -name "*.vm" -not -path "*/Tetris/*" -print -delete

.PHONY: all tar clean
```

6 Math.jack

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/12/Math.jack

/**
 * A basic math library.
 */
class Math {

    /** Initializes the library. */
    function void init() {
        return;
    }

    /** Returns the absolute value of x. */
    function int abs(int x) {
        if (x < 0) {
            let x = -x;
        }
        return x;
    }

    function int shiftLeft(int x, int shift) {
        while (shift > 0) {
            let x = x + x;
            let shift = shift - 1;
        }
        return x;
    }

    /** Returns the product of x and y. */
    function int multiply(int x, int y) {
        var int sum, shift, i, j;
        // Special Cases
        if (x = 1) {
            return y;
        }
        else {
            if (y = 1) {
                return x;
            }
            else {
                if (x = (-1)) {
                    return (-y);
                }
                else {
                    if (y = (-1)) {
                        return (-x);
                    }
                }
            }
        }
        if ((x = 0) | (y = 0)) {
            return 0;
        }
        // Powers of two:
        if (y = 2) {
            return x + x;
        }
        if (y = 4) {
            return (x + x) + (x + x);
        }
        if (y = 8) {
            return Math.shiftLeft(x, 3);
        }
        if (y = 16) {
            return Math.shiftLeft(x, 4);
        }
        if (y = 32) {
            return Math.shiftLeft(x, 5);
        }
        if (y = 64) {
            return Math.shiftLeft(x, 6);
        }
        if (y = 128) {
            return Math.shiftLeft(x, 7);
        }
        if (y = 256) {
            return Math.shiftLeft(x, 8);
        }
        let sum = 0;
        let shift = x;
        let i = 0;
        let j = 1;
        while (i < 16) {
            if (~(j & y = 0)) {
                let sum = sum + shift;
            }
            let shift = shift + shift;
            let j = j + j;
            let i = i + 1;
        }
        return sum;
    }

    /** Returns the integer part of x/y. */
    function int divide(int x, int y) {
        var int q, qy;
        // Check for division by zero
        if (y = 0) {
            do Sys.error(3);
            return -1;
        }
        // Check for negative numbers
        if ((x < 0) | (y < 0)) {
            if (~((x < 0) & (y < 0))) {
                // Only one is negative
                return -Math.divide(Math.abs(x), Math.abs(y));
            }
            else {
                // This means both are negative.
                let x = Math.abs(x);
                let y = Math.abs(y);
            }
        }
        // Special Cases
        if (y = 1) {
            return x;
        }
        if (y > x) {
            return 0;
        }

        if ((y + y) < 0) {
            // Overflow..
            let q = 0;
        }
        else {
            let q = Math.divide(x, y + y);
        }
        let qy = Math.multiply(q, y);
        if ((x - (qy + qy)) < y) {
            return (q + q);
        }
        else {
            return (q + q + 1);
        }
    }

    /** Returns the integer part of the square root of x. */
    function int sqrt(int x) {
        var int y, j, two2j, z;
        if (x < 0) {
            do Sys.error(4);
            return -1;
        }
        let y = 0;
        let z = 0;
        let j = 7;
        let two2j = 128;
        while (j > -1) {
            if ((y + two2j) < 182) {
                let z = y + two2j;
               if (((z * z) - 1) < x) {
                    let y = z;
                }
                if ((y * y) = x) {
                    return y;
                }
            }
            let two2j = two2j / 2;
            let j = j - 1;
        }
        return y;
    }

    /** Returns the greater number. */
    function int max(int a, int b) {
        var int x;
        let x = a;
        if (b > a) {
            let x = b;
        }
        return x;
    }

    /** Returns the smaller number. */
    function int min(int a, int b) {
        return -Math.max(-a, -b);
    }
}
```

7 Memory.jack

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/12/Memory.jack

/**
 * Memory operations library.
 */
class Memory {
    static Array freeList, base;
    static int len, next;

    /** Initializes memory parameters. */
    function void init() {
        let base = 0;
        let freeList = 2048;    // Heap base
        let len = 0;
        let next = 1;
        let freeList[len] = 14336;  // Heap size
        let freeList[next] = null;
        return;
    }

    /** Returns the value of the main memory at the given address. */
    function int peek(int address) {
        return base[address];
    }

    /** Sets the value of the main memory at this address
     *  to the given value. */
    function void poke(int address, int value) {
        let base[address] = value;
        return;
    }

    /** Swaps two entries in the freeList data structure */
    function void swap(Array curr, Array prev, Array prevprev) {
        var Array tmpNext;

        if (~(prevprev = null)) {
            let prevprev[next] = curr;
        }

        let tmpNext = prev;
        let prev[next] = curr[next];
        let curr[next] = tmpNext;
        return;
    }

    /** Sorts the freeList data structure */
    function void sort() {
        var Array curr, prev, prevprev, head;
        var int i, n;
        var boolean swapped;

        if (freeList = null) {
            return;
        }

        let swapped = true;
        let prevprev = null;
        let prev = freeList;
        let curr = freeList[next];
        let head = freeList;

        let n = 1;
        while (~(curr = null)) {
            let prevprev = prev;
            let prev = curr;
            let curr = curr[next];
            let n = n + 1;
        }
        while (swapped) {
            let swapped = false;
            let i = 1;
            let prevprev = null;
            let prev = head;
            let curr = prev[next];

            while (i < n) {
                if (curr < prev) {
                    if (i = 1) {
                        let head = curr;
                    }
                    do Memory.swap(curr, prev, prevprev);
                    let swapped = true;
                }
                let prevprev = prev;
                let prev = curr;
                let curr = curr[next];
                let i = i + 1;
            }
            let n = n - 1;
        }
        let freeList = head;
        return;
    }

    /** Performs defragmentation */
    function void defrag() {
        var Array prev, curr;

        do Memory.sort();
        let prev = freeList;
        let curr = prev[next];
        while (~(curr = null)) {
            //do Memory.printList();
            if ((prev + prev[len]) = curr) {
                let prev[next] = curr[next];
                let prev[len] = prev[len] + curr[len];
                let curr = curr[next];
            }
            else {
                let prev = curr;
                let curr = curr[next];
            }
        }
        return;
    }

    /** finds and allocates from the heap a memory block of the
     *  specified size and returns a reference to its base address. */
    function int alloc(int size) {
        var Array segment, prev, oldNext, output;
        var int diff;

        if (size < 1) {
            do Sys.error(5);
        }
        if (freeList = null) {
            do Sys.error(6); // Heap overflow
        }

        let prev = null;
        let segment = freeList;

        while ((size + 1) > segment[len]) {
            let prev = segment;
            let segment = segment[next];
            if ((segment = null)) {
                do Sys.error(6); // Even fragmentation didn't help!
            }
        }
        // This is reached when a free block was found
        let diff = segment[len] - size; // diff >= 1
        // If the found segment is large enough to fit more than
        // the required block, divide it to two blocks.
        if (diff > 3) {
            let output = segment + diff;
            let output[-1] = size + 1;
            let segment[len] = diff - 1;
        }
        else {
            if (prev = null) {
                let freeList = segment[next];
            }
            else {
                let prev[next] = segment[next];
            }
            let output = segment + 1;
        }
        //do Memory.printList();
        return output;
    }

    /** De-allocates the given object and frees its space. */
    function void deAlloc(int object) {
        var Array segment, curr, prev, nextSeg;

        let curr = freeList;
        let segment = object - 1;

        if (freeList = null) {
            let freeList = segment;
            let freeList[next] = null;
            return;
        }

        let nextSeg = freeList[next];

        while (~(nextSeg = null)) {
            let curr = nextSeg;
            let nextSeg = nextSeg[next];
        }
        let curr[next] = segment;
        let segment[next] = null;
        do Memory.defrag();
        return;
    }
}
```

8 Output.jack

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/12/Output.jack

/**
 * Handles writing characters to the screen.
 * The text screen (256 columns and 512 roes) is divided into 23 text rows (0..22),
 * each containing 64 text columns (0..63).
 * Each row is 11 pixels high (including 1 space pixel), and 8 pixels wide
 * (including 2 space pixels).
 */
class Output {

    // Character map for printing on the left of a screen word
    static Array charMaps;
    static int row, col, MAX_ROW, MAX_COL, SCREEN;

    /** Initializes the screen and locates the cursor at the screen's top-left. */
    function void init() {
        let row = 0;
        let col = 0;
        let MAX_ROW = 23;
        let MAX_COL = 64;
        let SCREEN = 16384;
        do Output.initMap();
        return;
    }

    // Initalizes the character map array
    function void initMap() {
        var int i;

        let charMaps = Array.new(127);

        // black square (used for non printable characters)
        do Output.create(0,63,63,63,63,63,63,63,63,63,0,0);

        // Assigns the bitmap for each character in the charachter set.
        do Output.create(32,0,0,0,0,0,0,0,0,0,0,0);          //
        do Output.create(33,12,30,30,30,12,12,0,12,12,0,0);  // !
        do Output.create(34,54,54,20,0,0,0,0,0,0,0,0);       // "
        do Output.create(35,0,18,18,63,18,18,63,18,18,0,0);  // #
        do Output.create(36,12,30,51,3,30,48,51,30,12,12,0); // £
        do Output.create(37,0,0,35,51,24,12,6,51,49,0,0);    // %
        do Output.create(38,12,30,30,12,54,27,27,27,54,0,0); // &
        do Output.create(39,12,12,6,0,0,0,0,0,0,0,0);        // '
        do Output.create(40,24,12,6,6,6,6,6,12,24,0,0);      // (
        do Output.create(41,6,12,24,24,24,24,24,12,6,0,0);   // )
        do Output.create(42,0,0,0,51,30,63,30,51,0,0,0);     // *
        do Output.create(43,0,0,0,12,12,63,12,12,0,0,0);     // +
        do Output.create(44,0,0,0,0,0,0,0,12,12,6,0);        // ,
        do Output.create(45,0,0,0,0,0,63,0,0,0,0,0);         // -
        do Output.create(46,0,0,0,0,0,0,0,12,12,0,0);        // .
        do Output.create(47,0,0,32,48,24,12,6,3,1,0,0);      // /

        do Output.create(48,12,30,51,51,51,51,51,30,12,0,0); // 0
        do Output.create(49,12,14,15,12,12,12,12,12,63,0,0); // 1
        do Output.create(50,30,51,48,24,12,6,3,51,63,0,0);   // 2
        do Output.create(51,30,51,48,48,28,48,48,51,30,0,0); // 3
        do Output.create(52,16,24,28,26,25,63,24,24,60,0,0); // 4
        do Output.create(53,63,3,3,31,48,48,48,51,30,0,0);   // 5
        do Output.create(54,28,6,3,3,31,51,51,51,30,0,0);    // 6
        do Output.create(55,63,49,48,48,24,12,12,12,12,0,0); // 7
        do Output.create(56,30,51,51,51,30,51,51,51,30,0,0); // 8
        do Output.create(57,30,51,51,51,62,48,48,24,14,0,0); // 9

        do Output.create(58,0,0,12,12,0,0,12,12,0,0,0);      // :
        do Output.create(59,0,0,12,12,0,0,12,12,6,0,0);      // ;
        do Output.create(60,0,0,24,12,6,3,6,12,24,0,0);      // <
        do Output.create(61,0,0,0,63,0,0,63,0,0,0,0);        // =
        do Output.create(62,0,0,3,6,12,24,12,6,3,0,0);       // >
        do Output.create(64,30,51,51,59,59,59,27,3,30,0,0);  // @
        do Output.create(63,30,51,51,24,12,12,0,12,12,0,0);  // ?

        do Output.create(65,12,30,51,51,63,51,51,51,51,0,0); // A ** TO BE FILLED **
        do Output.create(66,31,51,51,51,31,51,51,51,31,0,0); // B
        do Output.create(67,28,54,35,3,3,3,35,54,28,0,0);    // C
        do Output.create(68,15,27,51,51,51,51,51,27,15,0,0); // D
        do Output.create(69,63,51,35,11,15,11,35,51,63,0,0); // E
        do Output.create(70,63,51,35,11,15,11,3,3,3,0,0);    // F
        do Output.create(71,28,54,35,3,59,51,51,54,44,0,0);  // G
        do Output.create(72,51,51,51,51,63,51,51,51,51,0,0); // H
        do Output.create(73,30,12,12,12,12,12,12,12,30,0,0); // I
        do Output.create(74,60,24,24,24,24,24,27,27,14,0,0); // J
        do Output.create(75,51,51,51,27,15,27,51,51,51,0,0); // K
        do Output.create(76,3,3,3,3,3,3,35,51,63,0,0);       // L
        do Output.create(77,33,51,63,63,51,51,51,51,51,0,0); // M
        do Output.create(78,51,51,55,55,63,59,59,51,51,0,0); // N
        do Output.create(79,30,51,51,51,51,51,51,51,30,0,0); // O
        do Output.create(80,31,51,51,51,31,3,3,3,3,0,0);     // P
        do Output.create(81,30,51,51,51,51,51,63,59,30,48,0);// Q
        do Output.create(82,31,51,51,51,31,27,51,51,51,0,0); // R
        do Output.create(83,30,51,51,6,28,48,51,51,30,0,0);  // S
        do Output.create(84,63,63,45,12,12,12,12,12,30,0,0); // T
        do Output.create(85,51,51,51,51,51,51,51,51,30,0,0); // U
        do Output.create(86,51,51,51,51,51,30,30,12,12,0,0); // V
        do Output.create(87,51,51,51,51,51,63,63,63,18,0,0); // W
        do Output.create(88,51,51,30,30,12,30,30,51,51,0,0); // X
        do Output.create(89,51,51,51,51,30,12,12,12,30,0,0); // Y
        do Output.create(90,63,51,49,24,12,6,35,51,63,0,0);  // Z

        do Output.create(91,30,6,6,6,6,6,6,6,30,0,0);          // [
        do Output.create(92,0,0,1,3,6,12,24,48,32,0,0);        // \
        do Output.create(93,30,24,24,24,24,24,24,24,30,0,0);   // ]
        do Output.create(94,8,28,54,0,0,0,0,0,0,0,0);          // ^
        do Output.create(95,0,0,0,0,0,0,0,0,0,63,0);           // _
        do Output.create(96,6,12,24,0,0,0,0,0,0,0,0);          // `

        do Output.create(97,0,0,0,14,24,30,27,27,54,0,0);      // a
        do Output.create(98,3,3,3,15,27,51,51,51,30,0,0);      // b
        do Output.create(99,0,0,0,30,51,3,3,51,30,0,0);        // c
        do Output.create(100,48,48,48,60,54,51,51,51,30,0,0);  // d
        do Output.create(101,0,0,0,30,51,63,3,51,30,0,0);      // e
        do Output.create(102,28,54,38,6,15,6,6,6,15,0,0);      // f
        do Output.create(103,0,0,30,51,51,51,62,48,51,30,0);   // g
        do Output.create(104,3,3,3,27,55,51,51,51,51,0,0);     // h
        do Output.create(105,12,12,0,14,12,12,12,12,30,0,0);   // i
        do Output.create(106,48,48,0,56,48,48,48,48,51,30,0);  // j
        do Output.create(107,3,3,3,51,27,15,15,27,51,0,0);     // k
        do Output.create(108,14,12,12,12,12,12,12,12,30,0,0);  // l
        do Output.create(109,0,0,0,29,63,43,43,43,43,0,0);     // m
        do Output.create(110,0,0,0,29,51,51,51,51,51,0,0);     // n
        do Output.create(111,0,0,0,30,51,51,51,51,30,0,0);     // o
        do Output.create(112,0,0,0,30,51,51,51,31,3,3,0);      // p
        do Output.create(113,0,0,0,30,51,51,51,62,48,48,0);    // q
        do Output.create(114,0,0,0,29,55,51,3,3,7,0,0);        // r
        do Output.create(115,0,0,0,30,51,6,24,51,30,0,0);      // s
        do Output.create(116,4,6,6,15,6,6,6,54,28,0,0);        // t
        do Output.create(117,0,0,0,27,27,27,27,27,54,0,0);     // u
        do Output.create(118,0,0,0,51,51,51,51,30,12,0,0);     // v
        do Output.create(119,0,0,0,51,51,51,63,63,18,0,0);     // w
        do Output.create(120,0,0,0,51,30,12,12,30,51,0,0);     // x
        do Output.create(121,0,0,0,51,51,51,62,48,24,15,0);    // y
        do Output.create(122,0,0,0,63,27,12,6,51,63,0,0);      // z

        do Output.create(123,56,12,12,12,7,12,12,12,56,0,0);   // {
        do Output.create(124,12,12,12,12,12,12,12,12,12,0,0);  // |
        do Output.create(125,7,12,12,12,56,12,12,12,7,0,0);    // }
        do Output.create(126,38,45,25,0,0,0,0,0,0,0,0);        // ~

    return;
    }

    // Creates a character map array of the given char index with the given values.
    function void create(int index, int a, int b, int c, int d, int e,
                 int f, int g, int h, int i, int j, int k) {
    var Array map;

    let map = Array.new(11);
        let charMaps[index] = map;

        let map[0] = a;
        let map[1] = b;
        let map[2] = c;
        let map[3] = d;
        let map[4] = e;
        let map[5] = f;
        let map[6] = g;
        let map[7] = h;
        let map[8] = i;
        let map[9] = j;
        let map[10] = k;

        return;
    }

    // Returns the character map (array of size 11) for the given character
    // If an invalid character is given, returns the character map of a black square.
    function Array getMap(char c) {

        if ((c < 32) | (c > 126)) {
            let c = 0;
        }

        return charMaps[c];
    }

    /** Moves the cursor to the j'th column of the i'th row,
     *  and erases the character that was there. */
    function void moveCursor(int i, int j) {
        if ((i < 0) | (i > (MAX_ROW-1)) | (j < 0) | (j > (MAX_COL-1))) {
            do Sys.error(20);
            return;
        }
        let row = i;
        let col = j;
        do Output.drawChar(32);
        return;
    }

    function void drawChar(char c) {
        var Array map;
        var int address, charRow, currWord, mask, shift;
        var boolean firstInWord;

        let map = Output.getMap(c);
        let address = SCREEN + ((row * 352) + (col / 2)); // 32 * 11 = 352
        if ((col & 1) = 0) {
            let firstInWord = true;
        }
        else {
            let firstInWord = false;
        }

        let mask = 255;             // 0000000011111111
        let shift = 256;
        if (firstInWord) {
            let mask = -256;  // mask (shift 8)
            let shift = 1;
        }
        let charRow = 0;
        while (charRow < 11) {
            let currWord = Memory.peek(address);
            let currWord = (currWord & mask) | (map[charRow] * shift);
            do Memory.poke(address, currWord);
            let address = address + 32;
            let charRow = charRow + 1;
        }

        return;

    }

    function void incCursor() {
        let col = col + 1;
        if (col > (MAX_COL - 1)) {
            let col = 0;
            let row = row + 1;
        }
        if (row > (MAX_ROW - 1)) {
            let row = 0;
        }
        return;
    }

    /** Prints c at the cursor location and advances the cursor one
     *  column forward. */
    function void printChar(char c) {
        var Array map;
        var int address, charRow, currWord, mask, shift, firstInWord;

        // print new line
        if (c = String.newLine()) {
            do Output.println();
            return;
        }
        // print backspace
        if (c = String.backSpace()) {
            do Output.backSpace();
            return;
        }

        do Output.drawChar(c);

        do Output.incCursor();
        return;
    }

    /** Prints s starting at the cursor location, and advances the
     *  cursor appropriately. */
    function void printString(String s) {
        var int i, n;
        let i = 0;
        let n = s.length();
        while (i < n) {
            do Output.printChar(s.charAt(i));
            let i = i + 1;
        }
        return;
    }

    /** Prints i starting at the cursor location, and advances the
     *  cursor appropriately. */
    function void printInt(int i) {
        var String s;

        let s = String.new(10);
        do s.setInt(i);
        do Output.printString(s);

        return;
    }

    /** Advances the cursor to the beginning of the next line. */
    function void println() {
        let col = 0;
        let row = row + 1;
        if (row > (MAX_ROW-1)) {
            let row = 0;
        }
        return;
    }

    /** Moves the cursor one column back. */
    function void backSpace() {
        if (col = 0) {
            return;
        }
        do Output.moveCursor(row, col-1);
        return;
    }
}
```

9 Screen.jack

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/12/Screen.jack

/**
 * Graphic screen library.
 */
class Screen {
    static boolean color;
    static int SCREEN;
    static Array bits;

    /** Initializes the Screen. */
    function void init() {
        var int i, bit;
        let color = true;
        let SCREEN = 16384;

        let bits = Array.new(16);
        let i = 0;
        let bit = 1;
        while (i < 16) {
            let bits[i] = bit;
            let bit = bit + bit;
            let i = i + 1;
        }

        return;
    }

    /** Erases the whole screen. */
    function void clearScreen() {
        var int i;
        let i = 0;
        while (i < 8192) {
            do Memory.poke(SCREEN + i, 0);
            let i = i + 1;
        }
        return;
    }

    /** Sets the color to be used in further draw commands
     *  where white = false, black = true. */
    function void setColor(boolean b) {
        let color = b;
        return;
    }

    function boolean isValidPoint(int x, int y) {
        if ((x < 0) | (y < 0) | (x > 511) | (y > 255)) {
            return false;
        }
        return true;
    }

    function int div16(int x) {
        var int result, i;
        let i = 4;
        let result = 0;
        while (i < 16) {
            if (~((bits[i] & x)=0)) {
                let result = result + bits[i-4];
            }
            let i = i + 1;
        }
        return result;
    }

    /** Draws the (x, y) pixel. */
    function void drawPixel(int x, int y) {
        var int offset, xmod16, pixel, currWord;
        if (~Screen.isValidPoint(x, y)) {
            do Sys.error(7);
            return; // This isn't reached
        }
        let offset = SCREEN + ((y * 32) + Screen.div16(x));
        let xmod16 = x & 15;
        let pixel = bits[xmod16];
        let currWord = Memory.peek(offset);
        if (color) {
            let pixel = pixel | currWord;
        }
        else {
            let pixel = (~pixel) & currWord;
        }
        do Memory.poke(offset, pixel);
        return;
    }

    /** Draws a line from (x1, y1) to (x2, y2). */
    function void drawLine(int x1, int y1, int x2, int y2) {
        var int tmp, dx, dy, a, b, adyMinusbdx, inc, offset;

        if ((~Screen.isValidPoint(x1,y1)) | (~Screen.isValidPoint(x2,y2))) {
            do Sys.error(8);
            return;
        }

        let inc = 1;
        if (x1 = x2) {
            if (y1 < y2) {
                while (y1 < (y2 + 1)) {
                    do Screen.drawPixel(x1, y1);
                    let y1 = y1 + 1;
                }
                return;
            }
            else {
                while (y2 < (y1 + 1)) {
                    do Screen.drawPixel(x2, y2);
                    let y2 = y2 + 1;
                }
                return;
            }
        }
        if (y1 = y2) {
            if (x2 < x1) {
                let tmp = x2;
                let x2 = x1;
                let x1 = tmp;
            }
            while (x1 < (x2 + 1)) {
                let offset = SCREEN + ((y1 * 32) + Screen.div16(x1));
                if (((x1 & 15) = 0) & ((x1 + 15) < x2)) {
                    do Memory.poke(offset, color);
                    let x1 = x1 + 16;
                }
                else {
                    do Screen.drawPixel(x1, y1);
                    let x1 = x1 + 1;
                }
            }
                return;
        }
        if (x2 < x1) {
            let tmp = x1;
            let x1 = x2;
            let x2 = tmp;
            let tmp = y1;
            let y1 = y2;
            let y2 = tmp;
        }
        if (y2 < y1) {
            let inc = -1;
        }

        let dx = x2 - x1;
        let dy = Math.abs(y2 - y1);

        let a = 0;
        let b = 0;
        let adyMinusbdx = 0;

        while (~((a > dx) | (b > dy))) {
            do Screen.drawPixel(x1 + a, y1 + b);
            if (adyMinusbdx < 0) {
                let a = a + 1;
                let adyMinusbdx = adyMinusbdx + dy;
            }
            else {
                let b = b + inc;
                let adyMinusbdx = adyMinusbdx - dx;
            }
        }
        return;
    }

    /** Draws a filled rectangle where the top left corner
     *  is (x1, y1) and the bottom right corner is (x2, y2). */
    function void drawRectangle(int x1, int y1, int x2, int y2) {
        if ((~Screen.isValidPoint(x1,y1)) | (~Screen.isValidPoint(x2,y2)) |
            (x1 > x2) | (y1 > y2)) {
            do Sys.error(9);
            return;
        }
        while (y1 < (y2 + 1)) {
            do Screen.drawLine(x1,y1,x2,y1);
            let y1 = y1 + 1;
        }
        return;
    }

    /** Draws a filled circle of radius r around (cx, cy). */
    function void drawCircle(int cx, int cy, int r) {
        var int dy, dx;
        if (~Screen.isValidPoint(cx,cy)) {
            do Sys.error(12);
            return;
        }
        if ((~Screen.isValidPoint(cx + r,cy)) | (~Screen.isValidPoint(cx - r,cy)) |
            (~Screen.isValidPoint(cx,cy + r)) | (~Screen.isValidPoint(cx,cy - r)) |
            (r < 0)) {
            do Sys.error(13);
            return;
        }
        let dy = -r;
        while (dy < (r + 1)) {
            let dx = Math.sqrt((r*r) - (dy*dy));
            do Screen.drawLine(cx + dx, cy + dy, cx - dx, cy + dy);
            let dy = dy + 1;
        }
        return;
    }
}
```

10 String.jack

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/12/String.jack

/**
 * Represents a String object. Implements the String type.
 */
class String {
    field Array charsArr;
    field int size, maxSize;

    /** Constructs a new empty String with a maximum length of maxLength. */
    constructor String new(int maxLength) {
        if (maxLength < 0) {
            do Sys.error(14);
            return this;
        }
        if (maxLength > 0) {
            let charsArr = Array.new(maxLength);
        }
        let maxSize = maxLength;
        let size = 0;
        return this;
    }

    /** De-allocates the string and frees its space. */
    method void dispose() {
        if (maxSize > 0) {
            do charsArr.dispose();
        }
        return;
    }

    /** Returns the current length of this String. */
    method int length() {
        return size;
    }

    /** Returns the character at location j. */
    method char charAt(int j) {
        if ((j < 0) | (j > (size-1))) {
            do Sys.error(15);
            return 0;
        }
        return charsArr[j];
    }

    /** Sets the j'th character of this string to be c. */
    method void setCharAt(int j, char c) {
        if ((j < 0) | (j > (maxSize-1))) {
            do Sys.error(16);
            return;
        }
        let charsArr[j] = c;
        return;
    }

    /** Appends the character c to the end of this String.
```

```
     *  Returns this string as the return value. */
    method String appendChar(char c) {
        if (size = maxSize) {
            do Sys.error(17);
            return null;
        }
        do setCharAt(size, c);
        let size = size + 1;
        return this;
    }

    /** Erases the last character from this String. */
    method void eraseLastChar() {
        if (size = 0) {
            do Sys.error(18);
            return;
        }
        let size = size - 1;
        return;
    }

    /** Returns the integer value of this String until the first non
     *  numeric character. */
    method int intValue() {
        var int numSize, i, result, minSize;
        var boolean isNeg;
        let isNeg = false;
        let minSize = 0;
        let numSize = 0;
        let i = 1;
        let result = 0;

        if (size > 0) {
            if (charsArr[0] = 45) {
                let isNeg = true;
                let numSize = numSize + 1;
                let minSize = 1;
            }
        }

        while ((charsArr[numSize] > 47) & (charsArr[numSize] < 58)) {
            let numSize = numSize + 1;
        }
        while (numSize > minSize) {
            let numSize = numSize - 1;
            let result = result + ((charsArr[numSize] - 48) * i);
            let i = i * 10;
        }

        if (isNeg) {
            let result = -result;
        }

        return result;
    }

    /** Sets this String to hold a representation of the given number. */
    method void setInt(int number) {
        var int tmp, numlen;

        let numlen = 1;
        if (number = 0) {
            if (maxSize = 0) {
                do Sys.error(19);
                return;
            }
            let charsArr[0] = 48;
            return;
        }
        if (number < 0) {
            let charsArr[0] = 45; // Add '-' sign
            let numlen = numlen + 1;
            let number = -number;
        }
        let tmp = number;
        while ((tmp / 10) > 0) {
            let numlen = numlen + 1;
            let tmp = tmp / 10;
        }
        if (numlen > maxSize) {
            do Sys.error(19);
            return;
        }
        let size = numlen;
        let tmp = 0;
        while (number > 0) {
            // str[lastDigit] = number % 10
            let charsArr[numlen - tmp - 1] = (number - ((number / 10) * 10)) + 48;
            let number = number / 10;
            let tmp = tmp + 1;
        }

        return;
    }

    /** Returns the new line character. */
    function char newLine() {
        return 128;
    }

    /** Returns the backspace character. */
    function char backSpace() {
        return 129;
    }

    /** Returns the double quote (") character. */
    function char doubleQuote() {
        return 34;
    }
}
```

11 Sys.jack

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/12/Sys.jack

/**
 * A library of basic system services.
 */
class Sys {

    /** Performs all the initializations required by the OS. */
    function void init() {
        do Memory.init();
        do Math.init();
        do Screen.init();
        do Output.init();
        do Keyboard.init();
        do Main.main();
        do Sys.halt();
        return;
    }

    /** Halts execution. */
    function void halt() {
        while (true) {
        }
        return;
    }

    /** Waits approximately duration milliseconds and then returns. */
    function void wait(int duration) {
        var int i;

        if (duration < 0) {
            do Sys.error(1);
        }
        while (duration > 0) {
            let i = 50;
            while (i > 0) {
                let i = i - 1;
            }
            let duration = duration - 1;
        }
        return;
    }

    /** Prints the given error code in the form "ERR<errorCode>", and halts. */
    function void error(int errorCode) {
        do Output.printString("ERR");
        do Output.printInt(errorCode);
        do Sys.halt();
        return;
    }
}
```