Contents

1 Basic Test Results

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

  preparing sub.tar
dos2unix: converting file /tmp/bodek.A6A03O/nand2tet/Project12/roigreenberg/presubmission/testdir/stud/sub.tar/README to Uni
  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

```
roigreenberg,inbaravni

================================================================================

Roi Greenberg, ID 30557123, roi.greenberg@mail.huji.ac.il
Inbar Avni, ID 201131760, inbar.avni@mail.huji.ac.il

================================================================================



             Project 12 - Operating System

             ------------------------------




Submitted Files

---------------

README - This file.
Array.jack
Keyboard.jack
Math.jack
Memory.jack
Output.jack
Screen.jack
String.jack
Sys.jack


Remarks

-------

* No remarks for that time.
```

3 Array.jack

```
1  // This file is part of www.nand2tetris.org
2  // and the book "The Elements of Computing Systems"
3  // by Nisan and Schocken, MIT Press.
4  // File name: projects/12/Array.jack
5
6  /**
7   * Represents an array. Can be used to hold any type of object.
8   */
9  class Array {
10
11     /** Constructs a new Array of the given size. */
12     function Array new(int size) {
13         return Memory.alloc(size);
14     }
15
16     /** De-allocates the array and frees its space. */
17     method void dispose() {
18         do Memory.deAlloc(this);
19
20         return;
21     }
22 }
```

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;

    /** Initializes the keyboard. */
    function void init() {
        let keyboard  = 24576;
        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() {
        return Memory.peek(keyboard);
    }

    /**
     * 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 key, pKey;

        do Output.printChar(0);

        while (key = 0) {
            let key = Keyboard.keyPressed();
        }
        let pKey = key;
        while (~(key = 0)) {
            let key = Keyboard.keyPressed();
        }
```

```
60
61          do Output.backSpace();
62          do Output.printChar(pKey);
63
64          return pKey;
65      }
66
67      /**
68       * Prints the message on the screen, reads the next line
69       * (until a newline character) from the keyboard, and returns its value.
70       */
71      function String readLine(String message) {
72          var String s;
73          var char c;
74          var boolean t;
75          let t = true;
76          let s = String.new(64);
77
78          do Output.printString(message);
79
80          while(true) {
81              let c = Keyboard.readChar();
82              if(c = 128) {
83                  return s;
84              }
85              if(c = 129) {
86                  do s.eraseLastChar();
87
88              }
89              else {
90                  do s.appendChar(c);
91              }
92          }
93
94          return s;
95      }
96
97      /**
98       * Prints the message on the screen, reads the next line
99       * (until a newline character) from the keyboard, and returns its
100      * integer value (until the first non numeric character).
101      */
102     function int readInt(String message) {
103         var String s;
104
105         let s = Keyboard.readLine(message);
106
107         return s.intValue();
108     }
109 }
```

5 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 {
    static Array p_o_t;
    /** Initializes the library. */
    function void init() {
        var int i;
        let i = 0;
        let p_o_t = Array.new(16);
        let p_o_t[0] = 1;
        while(i < 15) {
            let i = i + 1;
            let p_o_t[i] = p_o_t[i - 1] + p_o_t[i - 1];
        }
        return;

    }

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

    /** Returns the product of x and y. */
    function int multiply(int x, int y) {
        var int sum;
        var int shiftX;
        var int j;

        let sum = 0;
        let shiftX = x;//Math.abs(x);
        let j = 0;

        while (j < 16) { // n . 15 maybe??
            if (Math.bit(y, j) = true) {
                let sum = sum + shiftX;
            }
            let shiftX = shiftX + shiftX;
            let j = j + 1;
        }
        /*if (((x < 0) & (y < 0)) | ((x > 0) & (y > 0))) {
        if (y < 0) {
            let sum = -sum;
        }*/
        return sum;
    }

    function int multiplyByTwo(int x, int j) {
        var int sum, i;
        let sum = x;
```

```
60          let i = 0;
61          while ( i < j){
62              let sum = sum + sum;
63              let i = i + 1;
64          }
65          return sum;
66      }
67
68      /** Returns the integer part of x/y. */
69      function int divide(int x, int y) {
70          var int q;
71          var int ax;
72          var int ay;
73
74
75
76          let ax = Math.abs(x);
77          let ay = Math.abs(y);
78
79          /*//do Output.printInt(x);
80          //do Output.printString("|");
81          //do Output.printInt(y);
82          //do Output.printString("||||");
83          //do Output.printInt(ax);
84          //do Output.printString("|");
85          //do Output.printInt(ay);
86
87          //do Output.println();
88          //do Output.printString("*****");
89          //do Output.println();
90  */
91          if (ay > ax) {
92              return 0;
93          }
94
95          if (((x < 0) & (y < 0)) | ((x > 0) & (y > 0))) {
96              return Math.abs_divide(ax, ay);
97          }
98          return -Math.abs_divide(ax, ay);
99      }
100
101     /** Returns the integer part of abs(x)/abs(y). */
102     function int abs_divide(int ax, int ay) {
103         var int q;
104         var int tmp;
105
106
107         if ((ay > ax) | (ay < 0)) {
108             return 0;
109         }
110
111         let q = Math.abs_divide(ax, ay + ay);
112         if (q = 0) {
113             return 1;
114         }
115         let tmp = q * ay;
116         if ((ax - (tmp + tmp)) < ay) {
117             return q + q;
118         } else {
119             return q + q + 1;
120         }
121     }
122
123     /** Returns the integer part of the square root of x. */
124     function int sqrt(int x) {
125         var int j;
126         var int two_power;
127         var int temp;
```

```
128             var int y;
129             let y = 0;
130             let j = 7; // n/2 -1
131
132             while (j > -1) {
133                 do Output.moveCursor(j, 0);
134                 let two_power = p_o_t[j];
135                 let temp = y + two_power;
136                 let temp = temp * temp;
137
138                 if ((temp > 0) & ~(temp > x)) {
139                     let y = y + two_power;
140                 }
141                 let j = j - 1;
142
143             }
144             return y;
145         }
146
147         /** Returns the greater number. */
148         function int max(int a, int b) {
149             if (a > b) {
150                 return a;
151             }
152             return b;
153
154         }
155
156         /** Returns the smaller number. */
157         function int min(int a, int b) {
158             if (a > b) {
159                 return b;
160             }
161             return a;
162         }
163
164         function boolean bit(int x, int j) {
165             return (~((p_o_t[j] & x) = 0));
166         }
167
168         function Array power_of_two() {
169             return (p_o_t);
170         }
171         function int mod(int x, int j) {
172             var int m;
173             let m = x;
174             while (m - j > 0){
175                 let m = m - j;
176             }
177             if (m = j){
178                 return 0;
179             }
180             return m;
181         }
182
183         function int mod16(int x) {
184             return x&15;
185         }
186     }
```

6 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 memory;
    static int freeList;
    static int next;

    /** Initializes memory parameters. */
    function void init() {
        let memory = 0; // maybe 0?
        let freeList = 2048;
        let next = 1;
        let memory[freeList] = 14333;
        let memory[freeList + next] = 0;
        return;
    }

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

    /** Sets the value of the main memory at this address
     *  to the given value. */
    function void poke(int address, int value) {
        let memory[address] = value;
        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 int addr;
        var int len;
        var int next_block;
        var int block;
        var boolean found;

        let found = false;
        let addr = freeList;

        while(~(found)) {
            let len = Memory.peek(addr);
            let next_block = Memory.peek(addr + next);

            if( len > size ) {
                let len = len - (size + 1);
                do Memory.poke(addr, len);
                let block = addr + len + 2;
                do Memory.poke(block - 1, size + 1);
                return block;

            }
```

```
60              let addr = next_block;
61
62              if (addr = 0) {
63                  do Sys.error(5);
64              }
65
66          }
67          return block;
68      }
69
70      /** De-allocates the given object and frees its space. */
71      function void deAlloc(int object) {
72          var int addr;
73          var int next_block;
74          var boolean found;
75
76          let addr = freeList;
77          let next_block = Memory.peek(addr + next);
78          let found = (next_block = 0);
79
80          while(~(found)) {
81              let addr = next_block;
82              let next_block = Memory.peek(addr + next);
83              let found = (next_block = 0);
84          }
85
86          do Memory.poke(addr + next, object - 1);
87          do Memory.poke(object - 1, Memory.peek(object - 1) - 2);
88          do Memory.poke(object, 0);
89
90          return;
91      }
92  }
```

7 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 {

    static int x, y;
    static Array screen;
    static int z;

    // Character map for printing on the left of a screen word
    static Array charMaps;

    /** Initializes the screen and locates the cursor at the screen's top-left. */
    function void init() {
        let x = 0;
        let y = 0;
        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(0,63,63,63,63,63,63,63,63,0,0);

        // Assigns the bitmap for each character in the charachter set.
        do Output.create(32,0,0,0,0,0,0,0,0,0,0);          //
        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);       // "
        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
```

```
60          do Output.create(49,12,14,15,12,12,12,12,12,63,0,0); // 1
61          do Output.create(50,30,51,48,24,12,6,3,51,63,0,0);   // 2
62          do Output.create(51,30,51,48,48,28,48,48,51,30,0,0); // 3
63          do Output.create(52,16,24,28,26,25,63,24,24,60,0,0); // 4
64          do Output.create(53,63,3,3,31,48,48,48,51,30,0,0);   // 5
65          do Output.create(54,28,6,3,3,31,51,51,51,30,0,0);    // 6
66          do Output.create(55,63,49,48,48,24,12,12,12,12,0,0); // 7
67          do Output.create(56,30,51,51,51,30,51,51,51,30,0,0); // 8
68          do Output.create(57,30,51,51,51,62,48,48,24,14,0,0); // 9
69
70          do Output.create(58,0,0,12,12,0,0,12,12,0,0,0);      // :
71          do Output.create(59,0,0,12,12,0,0,12,12,6,0,0);      // ;
72          do Output.create(60,0,0,24,12,6,3,6,12,24,0,0);      // <
73          do Output.create(61,0,0,0,63,0,0,63,0,0,0,0);        // =
74          do Output.create(62,0,0,3,6,12,24,12,6,3,0,0);       // >
75          do Output.create(64,30,51,51,59,59,59,27,3,30,0,0);  // @
76          do Output.create(63,30,51,51,24,12,12,0,12,12,0,0);  // ?
77
78          do Output.create(65,12,30,51,51,63,51,51,51,51,0,0); // A
79          do Output.create(66,31,51,51,51,31,51,51,51,31,0,0); // B
80          do Output.create(67,28,54,35,3,3,3,35,54,28,0,0);    // C
81          do Output.create(68,15,27,51,51,51,51,51,27,15,0,0); // D
82          do Output.create(69,63,51,35,11,15,11,35,51,63,0,0); // E
83          do Output.create(70,63,51,35,11,15,11,3,3,3,0,0);    // F
84          do Output.create(71,28,54,35,3,59,51,51,54,44,0,0);  // G
85          do Output.create(72,51,51,51,51,63,51,51,51,51,0,0); // H
86          do Output.create(73,30,12,12,12,12,12,12,12,30,0,0); // I
87          do Output.create(74,60,24,24,24,24,24,27,27,14,0,0); // J
88          do Output.create(75,51,51,51,27,15,27,51,51,51,0,0); // K
89          do Output.create(76,3,3,3,3,3,3,35,51,63,0,0);       // L
90          do Output.create(77,33,51,63,63,51,51,51,51,51,0,0); // M
91          do Output.create(78,51,51,55,55,63,59,59,51,51,0,0); // N
92          do Output.create(79,30,51,51,51,51,51,51,51,30,0,0); // O
93          do Output.create(80,31,51,51,51,31,3,3,3,3,0,0);     // P
94          do Output.create(81,30,51,51,51,51,51,63,59,30,48,0);// Q
95          do Output.create(82,31,51,51,51,51,31,27,51,51,51,0,0); // R
96          do Output.create(83,30,51,51,6,28,48,51,51,30,0,0);  // S
97          do Output.create(84,63,63,45,12,12,12,12,12,30,0,0); // T
98          do Output.create(85,51,51,51,51,51,51,51,51,30,0,0); // U
99          do Output.create(86,51,51,51,51,51,30,30,12,12,0,0); // V
100         do Output.create(87,51,51,51,51,51,63,63,63,18,0,0); // W
101         do Output.create(88,51,51,30,30,12,30,30,51,51,0,0); // X
102         do Output.create(89,51,51,51,51,30,12,12,12,30,0,0); // Y
103         do Output.create(90,63,51,49,24,12,6,35,51,63,0,0);  // Z
104
105         do Output.create(91,30,6,6,6,6,6,6,6,30,0,0);          // [
106         do Output.create(92,0,0,1,3,6,12,24,48,32,0,0);        // \
107         do Output.create(93,30,24,24,24,24,24,24,24,30,0,0);   // ]
108         do Output.create(94,8,28,54,0,0,0,0,0,0,0,0);          // ^
109         do Output.create(95,0,0,0,0,0,0,0,0,0,63,0);           // _
110         do Output.create(96,6,12,24,0,0,0,0,0,0,0,0);          // `
111
112         do Output.create(97,0,0,0,14,24,30,27,27,54,0,0);      // a
113         do Output.create(98,3,3,3,15,27,51,51,51,30,0,0);      // b
114         do Output.create(99,0,0,0,30,51,3,3,51,30,0,0);        // c
115         do Output.create(100,48,48,48,60,54,51,51,51,30,0,0);  // d
116         do Output.create(101,0,0,0,30,51,63,3,51,30,0,0);      // e
117         do Output.create(102,28,54,38,6,15,6,6,6,15,0,0);      // f
118         do Output.create(103,0,0,30,51,51,51,62,48,51,30,0);   // g
119         do Output.create(104,3,3,3,27,55,51,51,51,51,0,0);     // h
120         do Output.create(105,12,12,0,14,12,12,12,12,30,0,0);   // i
121         do Output.create(106,48,48,0,56,48,48,48,48,51,30,0);  // j
122         do Output.create(107,3,3,3,51,27,15,15,27,51,0,0);     // k
123         do Output.create(108,14,12,12,12,12,12,12,12,30,0,0);  // l
124         do Output.create(109,0,0,0,29,63,43,43,43,43,0,0);     // m
125         do Output.create(110,0,0,0,29,51,51,51,51,51,0,0);     // n
126         do Output.create(111,0,0,0,30,51,51,51,51,30,0,0);     // o
127         do Output.create(112,0,0,0,30,51,51,51,31,3,3,0);      // p
```

```
128         do Output.create(113,0,0,0,30,51,51,51,62,48,48,0);    // q
129         do Output.create(114,0,0,0,29,55,51,3,3,7,0,0);        // r
130         do Output.create(115,0,0,0,30,51,6,24,51,30,0,0);      // s
131         do Output.create(116,4,6,6,15,6,6,6,54,28,0,0);        // t
132         do Output.create(117,0,0,0,27,27,27,27,27,54,0,0);     // u
133         do Output.create(118,0,0,0,51,51,51,51,30,12,0,0);     // v
134         do Output.create(119,0,0,0,51,51,51,63,63,18,0,0);     // w
135         do Output.create(120,0,0,0,51,30,12,12,30,51,0,0);     // x
136         do Output.create(121,0,0,0,51,51,51,62,48,24,15,0);    // y
137         do Output.create(122,0,0,0,63,27,12,6,51,63,0,0);      // z
138
139         do Output.create(123,56,12,12,12,7,12,12,12,56,0,0);   // {
140         do Output.create(124,12,12,12,12,12,12,12,12,12,0,0); // |
141         do Output.create(125,7,12,12,12,56,12,12,12,7,0,0);    // }
142         do Output.create(126,38,45,25,0,0,0,0,0,0,0,0);        // ~
143
144     return;
145     }
146
147     // Creates a character map array of the given char index with the given values.
148     function void create(int index, int a, int b, int c, int d, int e,
149                  int f, int g, int h, int i, int j, int k) {
150         var Array map;
151
152         let map = Array.new(11);
153         let charMaps[index] = map;
154
155         let map[0] = a;
156         let map[1] = b;
157         let map[2] = c;
158         let map[3] = d;
159         let map[4] = e;
160         let map[5] = f;
161         let map[6] = g;
162         let map[7] = h;
163         let map[8] = i;
164         let map[9] = j;
165         let map[10] = k;
166
167         return;
168     }
169
170     // Returns the character map (array of size 11) for the given character
171     // If an invalid character is given, returns the character map of a black square.
172     function Array getMap(char c) {
173
174         if ((c < 32) | (c > 126)) {
175             let c = 0;
176         }
177
178         return charMaps[c];
179     }
180
181     /** Moves the cursor to the jth column of the ith row,
182      *  and erases the character that was there. */
183     function void moveCursor(int i, int j) {
184         let x = j;
185         let y = i;
186         do Output.doPrintChar(32, false);
187         return;
188     }
189
190     /** Prints c at the cursor location and advances the cursor one
191      *  column forward. */
192     function void printChar(char c) {
193         if(c = 128) {
194             do Output.println();
195             return;
```

```
196         }
197         if(c = 129) {
198             do Output.backSpace();
199             return;
200         }
201         do Output.doPrintChar(c, true);
202         return;
203     }
204 
205     function void doPrintChar(char c, boolean advance) {
206         var Array map;
207         var int index, pos;
208 
209         let index = 0;
210         let pos = (Math.multiplyByTwo(y , 5) * 11) + (x / 2);
211 
212         let map = Output.getMap(c);
213         while ( index < 11 ) {
214         if (Math.mod(x,2) = 1) {
215             let screen[pos] = (screen[pos] & 255) | (map[index] * 256);
216         }
217         else {
218             let z = 100;
219             let screen[pos] = (screen[pos] & -256) | (map[index]);
220         }
221 
222 
223             let pos = pos + 32;
224             let index = index + 1;
225         }
226         if (advance) {
227             if (x < 63 ){
228                 let x = x + 1;
229             } else {
230                 do Output.println();
231             }
232         }
233         return;
234     }
235 
236     /** Prints s starting at the cursor location, and advances the
237      *  cursor appropriately. */
238     function void printString(String s) {
239         var int i, l;
240         var char c;
241 
242         let i = 0;
243         let l = s.length();
244         while (i < l) {
245             let c = s.charAt(i);
246             do Output.printChar(c);
247             let i = i + 1;
248         }
249         return;
250     }
251 
252     /** Prints i starting at the cursor location, and advances the
253      *  cursor appropriately. */
254     function void printInt(int i) {
255         var String s;
256         let s = String.new(6);
257         do s.setInt(i);
258         do Output.printString(s);
259         do s.dispose();
260         return;
261     }
262 
263     /** Advances the cursor to the beginning of the next line. */
```

```
264       function void println() {
265           if (y < 22 ){
266               do Output.moveCursor(y + 1, 0);
267           } else {
268               do Output.moveCursor(0, 0);
269           }
270
271           return;
272       }
273
274       /** Moves the cursor one column back. */
275       function void backSpace() {
276           if ( x = 0 ) {
277               if ( y = 0 ) {
278                   do Output.moveCursor(22, 63);
279               }
280               else {
281                   do Output.moveCursor(y - 1, 63);
282               }
283           }
284           else {
285               do Output.moveCursor(y, x - 1);
286           }
287
288       return;
289       }
290   }
```

8 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 currColor;
    static Array bitIndex;

    /** Initializes the Screen. */
    function void init() {

        let bitIndex = Math.power_of_two();
        let currColor = true;

        return;
    }

    /** Erases the whole screen. */
    function void clearScreen() {

        do Screen.setColor(false);
        do Screen.drawRectangle(0, 0, 511, 255);
        do Screen.setColor(true);

        return;

    }

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

        let currColor = b;
        return;

    }

    /** function void drawPixel(int x, int y) {
        var int, pos, screenLoc, address, val;
        let screenLoc = 16384;
        let pos = screenLoc + (y * 32) + (x / 16);
        let val = Memory.peek(pos);

        //draw black pixel
        if (currColor) {
            let val = -1;

        //draw white pixel
        } else {
            let val = 0;
        }

        // set the pixel
        do Memory.poke(pos, val);
```

```
60
61          return;
62      } */
63
64      /** Draws the (x, y) pixel. */
65      function void drawPixel(int x, int y) {
66
67          var int bitInPos, pos, screenLoc, address, val;
68          let bitInPos = Math.mod16(x); // col % 16
69          let screenLoc = 16384;
70          let pos = screenLoc + Math.multiplyByTwo(y , 5) + (x / 16);
71          let val = Memory.peek(pos);
72
73          //draw black pixel
74          if (currColor) {
75              let val = val | bitIndex[bitInPos];
76
77          //draw white pixel
78          } else {
79              let val = val & ~bitIndex[bitInPos];
80          }
81
82          // set the pixel
83          do Memory.poke(pos, val);
84
85          return;
86
87      }
88
89      function void drawHLine(int x1, int x2, int y, int pos) {
90          //var int pos;
91
92          if (Math.mod16(x1) > 0) {
93              let pos = pos + 1;
94          }
95
96          while ((~(x1 > x2)) & (Math.mod16(x1) > 0)){
97              do Screen.drawPixel(x1, y);
98              let x1 = x1 + 1;
99          }
100
101         //let pos = 16384 + Math.multiplyByTwo(y , 5) + (x1 / 16);
102
103         while (~(x1 + 16 > x2)){
104             //draw black pixel
105             if (currColor) {
106                 do Memory.poke(pos, -1);
107
108             //draw white pixel
109             } else {
110                 do Memory.poke(pos, 0);
111             }
112             let pos = pos + 1;
113             let x1 = x1 + 16;
114         }
115
116         while (~(x1 > x2)){
117             do Screen.drawPixel(x1, y);
118             let x1 = x1 + 1;
119         }
120
121         return;
122     }
123
124     /** Draws a line from (x1, y1) to (x2, y2). */
125     function void drawLine(int x1, int y1, int x2, int y2) {
126
127         var int tmp, fillGapX, fillGapY, gapX, gapY, directY;
```

```
128
129         if (~(x1 < x2)) {
130
131             let tmp = x1;
132             let x1 = x2;
133             let x2 = tmp;
134
135             let tmp = y1;
136             let y1 = y2;
137             let y2 = tmp;
138         }
139         do Screen.drawPixel(x1, y1);
140
141         let fillGapX = 0;
142         let fillGapY = 0;
143         let gapY = y2 - y1;
144         let gapX = x2 - x1;
145
146
147         if (gapY = 0){
148             do Screen.drawHLine(x1, x2, y1, 16384 + Math.multiplyByTwo(y1 , 5) + (x1 / 16));
149             /*while (~(fillGapX = gapX)){
150                 let fillGapX = fillGapX + 1;
151                 do Screen.drawPixel(x1 + fillGapX, y1);
152             }*/
153             return;
154         }
155
156
157         if (gapX = 0){
158
159             if (gapY < 0){
160                 let directY = -1;
161             } else {
162                 let directY = 1;
163             }
164
165             while (~(fillGapY = gapY)){
166                 let fillGapY = fillGapY + directY;
167                 do Screen.drawPixel(x1, y1 + fillGapY);
168             }
169             return;
170         }
171
172         /** while ((fillGapX = gapX) & (~(fillGapY = gapY))){
173
174             if (((fillGapX * gapY) - (fillGapY * gapX)) < 0){
175                 let fillGapX = fillGapX + 1;
176             } else {
177                 let fillGapY = fillGapY + directY;
178             }
179             do Screen.drawPixel(x1 + fillGapX, y1 + fillGapY);
180
181         } */
182
183         while ((fillGapX < gapX) & (fillGapY < gapY)){
184
185             if (((fillGapX * gapY) - (fillGapY * gapX)) < 0){
186                 let fillGapX = fillGapX + 1;
187             } else {
188                 let fillGapY = fillGapY + 1;
189             }
190             do Screen.drawPixel(x1 + fillGapX, y1 + fillGapY);
191
192         }
193
194         while ((fillGapX < gapX) & (fillGapY > gapY)){
195
```

```
            if (((fillGapX * gapY) - (fillGapY * gapX)) > 0){
                let fillGapX = fillGapX + 1;
            } else {
                let fillGapY = fillGapY - 1;
            }
            do Screen.drawPixel(x1 + fillGapX, y1 + fillGapY);

        }


        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) {

        var int gapX, gapY, fillGapX, fillGapY, pos;

        let gapX = x2 - x1;
        let gapY = y2 - y1;
        let fillGapX = 0;
        let fillGapY = 0;
        let pos = 16384 + (y1 * 32) + (x1 / 16);
        while (~(fillGapY > gapY)){
            //do Screen.drawLine(x1, y1 + fillGapY, x2, y1 + fillGapY);
            do Screen.drawHLine(x1, x2, y1 + fillGapY, pos);
            let pos = pos + 32;
            let fillGapY = fillGapY + 1;
        }


        return;
    }

    /** Draws a filled circle of radius r around (cx, cy). */
    function void drawCircle(int cx, int cy, int r) {

        var int gapX, gapY, sqrt, cx_minus, cx_plus, cy_minus, cy_plus;

        let gapY = -r;
        while(~(gapY = 0)){
            let sqrt = Math.sqrt((r * r) - (gapY * gapY));
            let cx_minus = cx - sqrt;
            let cx_plus = cx + sqrt;
            let cy_minus = cy - gapY;
            let cy_plus = cy + gapY;
            let gapY = gapY + 1;
            do Screen.drawLine(cx_minus, cy_minus, cx_plus, cy_minus);
            do Screen.drawLine(cx_minus, cy_plus, cx_plus, cy_plus);
        }
        do Screen.drawLine(cx - r, cy, cx + r, cy);
        return;
    }
}
```

9 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 strArr;
    field int strMaxSize;
    field int strSize;

    /** Constructs a new empty String with a maximum length of maxLength. */
    constructor String new(int maxLength) {

        let strMaxSize = maxLength;
        let strArr = Array.new(strMaxSize);
        let strSize = 0;

        return this;
    }

    /** De-allocates the string and frees its space. */
    method void dispose() {

        do Memory.deAlloc(this);
        return;
    }

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

    /** Returns the character at location j. */
    method char charAt(int j) {
        return strArr[j];
    }

    /** Sets the j'th character of this string to be c. */
    method void setCharAt(int j, char c) {

        let strArr[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) {
let     strArr[strSize] = c;
        let strSize = strSize + 1;
       /* var int tmpSize;
        let tmpSize = strSize + 1;
        if (~(tmpSize = strMaxSize)){
            let strArr[strSize] = c;
            let strSize = strSize + 1;

```

```
60          }*/
61
62          return this;
63      }
64
65      /** Erases the last character from this String. */
66      method void eraseLastChar() {
67
68          if (strSize > 0){
69              let strSize = strSize - 1;
70          }
71          return;
72      }
73
74      /** Returns the integer value of this String until the first non
75       *  numeric character. */
76      method int intValue() {
77
78          var int digVal;
79          var boolean isNeg;
80          var int index;
81
82          //check if negative
83          if (strSize > 0){
84              if (strArr[0] = 45){
85                  let isNeg = true;
86                  let index = 1;
87              }
88              let isNeg = false;
89              let index = 0;
90          }
91
92          while ((index < strSize) & (String.isNumericChar(strArr[index]))){
93              let digVal = (10 * digVal) + (String.digitValOfChar(strArr[index]));
94              let index = index + 1;
95          }
96
97          if (~(isNeg)){
98              return digVal;
99          }
100
101         return -digVal;
102     }
103
104     /** Sets this String to hold a representation of the given number. */
105     method void setInt(int number) {
106         var int divByTen;
107         var int modulu;
108         var int length, i, j;
109         var boolean neg;
110         var char digAsChar, tmp;
111
112
113         let strSize = 0;
114         let length = 0;
115         let neg = false;
116
117         if (number = 0){
118             do appendChar(48);
119             return;
120         }
121
122         if (number < 0){
123             let number = -number;
124             do appendChar(45);
125             let neg = true;
126         }
127
```

```
128
129
130         while (number > 0) {
131             let length = length  + 1;
132             let modulu = Math.mod(number,10) ;
133             let digAsChar = String.digitInAscii(modulu);
134             do appendChar(digAsChar);
135             let number = number / 10;
136         }
137
138         let i = strSize - length;
139         let j = strSize - 1;
140         while ( i < j ) {
141             let tmp = strArr[i];
142             let strArr[i] = strArr[j];
143             let strArr[j] = tmp;
144             let i = i + 1;
145             let j = j - 1;
146
147         }
148
149
150         return;
151     }
152
153
154     method void recursiveSetInt(int number){
155
156         var int divByTen;
157         var int modulu;
158         var char digAsChar;
159
160         while (number > 0) {
161
162             let modulu = Math.mod(number,10) ;
163             let digAsChar = String.digitInAscii(modulu);
164             do appendChar(digAsChar);
165             let number = number / 10;
166
167         }
168        /* let divByTen = number / 10;
169         let modulu = Math.mod(number,10) ;
170         let digAsChar = String.digitInAscii(modulu);
171
172         if (number < 10){
173             do appendChar(digAsChar);
174         } else {
175             do recursiveSetInt(divByTen);
176             do appendChar(digAsChar);
177         }*/
178         return;
179     }
180
181     /** Returns the new line character. */
182     function char newLine() {
183         return 128;
184     }
185
186     /** Returns the backspace character. */
187     function char backSpace() {
188         return 129;
189     }
190
191     /** Returns the double quote (") character. */
192     function char doubleQuote() {
193         return 34;
194     }
195
```

```
196     function boolean isNumericChar(char digChar){
197         if (~(digChar < 48)){
198             if (~(digChar > 57)){
199                 return true;
200             }
201             return false;
202         }
203         return false;
204     }
205 
206     function int digitValOfChar(char digChar){
207         return (digChar - 48);
208     }
209 
210     function char digitInAscii(int dig){
211         return (48 + dig);
212     }
213 
214 }
```

10 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 Output.init();
        do Screen.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;
        while( duration > 0 ) {
            let i = 0;
            while( i < 130 ) {
                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("Error: ");
        do Output.printInt(errorCode);
        do Sys.halt();
        return;
    }
}
```