Contents

1 Basic Test Results

```
********** TESTING FOLDER STRUCTURE START **********
Running test12.sh:
Your logins are: muaz.abdeen, is that ok?

********** TESTING FOLDER STRUCTURE END **********

********** PROJECT TEST START **********

There are no presubmission tests for this project,
so test it yourself - it's more fun that way!
See you on the test ;)

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

2 README

```
muaz.abdeen
================================================================================
Muaz Abdeen, ID 300575297, muaz.abdeen@mail.huji.ac.il
================================================================================

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


Submitted Files
---------------
(1)  README                 - This file.
(2)  Array.jack             - Represents an array.
(3)  Keyboard.jack          - Handling user input from the keyboard.
(4)  Math.jack              - A basic math library.
(5)  Memory.jack            - Memory operations library.
(6)  Output.jack            - Handles writing characters to the screen.
(7)  Screen.jack            - Graphic screen library.
(8)  String.jack            - Represents a String object.
(9)  Sys.jack               - A library of basic system services.


Remarks
-------
* ...

```

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.
8   * In the Jack language, arrays are instances of the Array class.
9   * Once declared, the array entries can be accessed using the usual
10  * syntax arr[i]. Each array entry can hold a primitive data type as
11  * well as any object type. Different array entries can have different
12  * data types.
13  */
14 class Array {
15
16     /** Constructs a new Array of the given size. */
17     function Array new(int size) {
18         return Memory.alloc(size);
19     }
20
21     /** Disposes this array. */
22     method void dispose() {
23         do Memory.deAlloc(this);
24         return;
25     }
26 }
```

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 {

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

    /**
     * Returns the character of the currently pressed key on the keyboard;
     * if no key is currently pressed, returns 0.
     *
     * Recognizes all ASCII characters, as well as the following 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(24576);
    }

    /**
     * Waits until a key is pressed on the keyboard and released,
     * then echoes the key to the screen, and returns the character
     * of the pressed key.
     */
    function char readChar() {
        var char character;
        do Output.printChar(0);
        while (Keyboard.keyPressed() = 0) {}
        let character = Keyboard.keyPressed();
        while (Keyboard.keyPressed() > 0) {}
        do Output.backSpace();
        do Output.printChar(character);
        return character;
    }

    /**
     * Displays the message on the screen, reads from the keyboard the entered
     * text until a newline character is detected, echoes the text to the screen,
     * and returns its value. Also handles user backspaces.
```

```
60        */
61       function String readLine(String message) {
62           var char c, newLine, backSpace;
63           var String buffer;
64           var boolean endOfLine;
65           let buffer = String.new(80);
66           do Output.printString(message);
67
68           let newLine = String.newLine();
69           let backSpace = String.backSpace();
70
71           while (~endOfLine) {
72               let c = Keyboard.readChar();
73               let endOfLine = (c = newLine);
74               if (~endOfLine) {
75                   if (c = backSpace) {
76                       do buffer.eraseLastChar();
77                   } else {
78                       let buffer = buffer.appendChar(c);
79                   }
80               }
81           }
82           return buffer;
83       }
84
85       /**
86        * Displays the message on the screen, reads from the keyboard the entered
87        * text until a newline character is detected, echoes the text to the screen,
88        * and returns its integer value (until the first non-digit character in the
89        * entered text is detected). Also handles user backspaces.
90        */
91       function int readInt(String message) {
92           var String line;
93           var int intValue;
94           let line = Keyboard.readLine(message);
95           let intValue = line.intValue();
96           do line.dispose();
97           return intValue;
98       }
99   }
```

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 library of commonly used mathematical functions.
 * Note: Jack compilers implement multiplication and division using OS method calls.
 */
class Math {

    static int _2qy;
    static Array twoToThe;

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

    /** Returns true if the i-th bit of x is 1, false otherwise */
    function boolean bit(int x, int i) {
        // ~((x & twoToThe[i]) = 0);
        return x & twoToThe[i] > 0;
    }

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

    /** Returns the product of x and y.
     *  When a Jack compiler detects the multiplication operator '*' in the
     *  program's code, it handles it by invoking this method. In other words,
     *  the Jack expressions x*y and multiply(x,y) return the same value.
     */
    function int multiply(int x, int y) {
        var int sum, temp, twoPowSum, j;
        var boolean diffSigns;
        let diffSigns = (x < 0 & y > 0) | (x > 0 & y < 0);
        // remove the signs
        let x = Math.abs(x);
        let y = Math.abs(y);
        /* if the multiplicand smaller than the multiplier,
         * then swap them. */
        if (x < y) {
            let temp = x;
            let x = y;
            let y = temp;
```

```
60          }
61          // loop times of the multiplicand bits
62          while ((twoPowSum-1) < (y-1)) {
63              if (Math.bit(y, j)) {
64                  let sum = sum + x;
65                  let twoPowSum = twoToThe[j] + twoPowSum;
66              }
67              let x = x + x;
68              let j = j + 1;
69          }
70          // if signs of the operands are different
71          if (diffSigns) {
72              let sum = -sum;
73          }
74          return sum;
75      }
76
77      /** Returns the integer part of x/y.
78       *  When a Jack compiler detects the division operator '/' in the
79       *  program's code, it handles it by invoking this method. In other words,
80       *  the Jack expressions x/y and divide(x,y) return the same value.
81       */
82      function int divide(int x, int y) {
83          let _2qy = 0;
84          return Math.helperDivide(x, y);
85      }
86
87      function int helperDivide(int x, int y) {
88          var int res;
89          var boolean diffSigns;
90          if (y=0) {
91              do Sys.error("zeroDivisionError");
92              return 0;
93          }
94          let diffSigns = (x < 0 & y > 0) | (x > 0 & y < 0);
95          // remove the signs
96          let x = Math.abs(x);
97          let y = Math.abs(y);
98          if (y > x) {
99              return 0;
100         }
101         let res = Math.divide(x, y+y);
102         if (x - _2qy < y) {
103             let res = res + res;
104         } else {
105             let _2qy = _2qy + y;
106             let res = res + res + 1;
107         }
108         // if signs of the operands are different
109         if (diffSigns) {
110             let res = -res;
111         }
112         return res;
113     }
114
115     /** Returns the integer part of the square root of x. */
116     function int sqrt(int x) {
117         var int result;
118         var int j;
119         var int checked;
120         var int squared;
121
122         let result = 0;
123         let j = 7;
124         while(~(j < 0)) {
125             let checked = result + twoToThe[j];
126             let squared = Math.multiply(checked, checked);
127             if(~(squared > x) & (squared > 0)) {
```

```
                let result = checked;
            }
            let j = j - 1;
        }
        return result;
    }

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

    /** Returns the smaller number. */
    function int min(int a, int b) {
        if (a < b) {
            return a;
        } else {
            return b;
        }
    }
}
```

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

/**
 * This library provides two services: direct access to the computer's main
 * memory (RAM), and allocation and recycling of memory blocks. The Hack RAM
 * consists of 32,768 words, each holding a 16-bit binary number.
 */
class Memory {
    static Array ram;
    static Array heap;
    static int freeList;

    /** Initializes the class. */
    function void init() {
      let ram = 0;
      let heap = 2048;
      let freeList = 0;
      let heap[0] = 0;        // next
      let heap[1] = 14335;    // length
      return;
    }

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

    /** Sets the RAM value at the given address to the given value. */
    function void poke(int address, int value) {
      let ram[address] = value;
      return;
    }

    /** Finds an available RAM block of the given size and returns
     *  a reference to its base address. */
    function int alloc(int size) {
      var int next;
      let next = heap[freeList];
      while (heap[next+1] < (size + 2)) {
        let next = heap[next];
      }
      // update freeList size after chopping
      let heap[next+1] = heap[next+1] - (size+1);

      let heap[heap[next+1] + 1] = size + 1;
      return heap[next+1] + 2;
    }

    /** De-allocates the given object (cast as an array) by making
     *  it available for future allocations. */
    function void deAlloc(Array o) {
      var int next;
      var int ptr;
      let next = 0;
      while(~(heap[next] = 0)) {
        let next = heap[next];
```

```
60          }
61          let ptr = o-1;
62          let heap[ptr] = 0;
63          let heap[next] = ptr;
64
65          return;
66        }
67    }
```

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

/**
 * A library of functions for writing text on the screen.
 * The Hack physical screen consists of 512 rows of 256 pixels each.
 * The library uses a fixed font, in which each character is displayed
 * within a frame which is 11 pixels high (including 1 pixel for inter-line
 * spacing) and 8 pixels wide (including 2 pixels for inter-character spacing).
 * The resulting grid accommodates 23 rows (indexed 0..22, top to bottom)
 * of 64 characters each (indexed 0..63, left to right). The top left
 * character position on the screen is indexed (0,0). A cursor, implemented
 * as a small filled square, indicates where the next character will be displayed.
 */
class Output {

    // Character map for displaying characters
    static Array charMaps;
    static int row, col;

    /** Initializes the screen, and locates the cursor at the screen's top-left. */
    function void init() {
        do Output.initMap();
        do Output.moveCursor(0,0);
        return;
    }

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

        let charMaps = Array.new(127);

        // Black square, used for displaying 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.
        // The first parameter is the character index, the next 11 numbers
        // are the values of each row in the frame that represents this character.
        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
```

```
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 ** TO BE FILLED **
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 the character map array of the given character index, using 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) of the given character.
171     // If the given character is invalid or non-printable, returns the
172     // character map of a black square.
173     function Array getMap(char c) {
174         if ((c < 32) | (c > 126)) {
175             let c = 0;
176         }
177         return charMaps[c];
178     }
179
180     /** Moves the cursor to the j-th column of the i-th row,
181      *  and erases the character displayed there. */
182     function void moveCursor(int i, int j) {
183         var int k;
184         var int cursorX, cursorY;
185         var int mask, coefficient;
186         var int address;
187         var int temp;
188
189         let row = i;
190         let col = j;
191         let cursorX = (col/2);
192         let cursorY = row*11;
193         if((col&1)=0) {
194             let coefficient = 1;
195             let mask=255;
```

```
            let mask=~mask;
        } else {
            let coefficient=256;
            let mask=255;
        }

        while(k < 11){
            let address = 16384+(cursorY*32)+cursorX;
            let temp = Memory.peek(address);
            let temp =(temp&mask) + (0*coefficient);
            do Memory.poke(address,temp);
            let cursorY = cursorY+1;
            let k = k+1;
        }
        return;
    }

    /** Displays the given character at the cursor location,
     *  and advances the cursor one column forward. */
    function void printChar(char c) {
        var int k;
        var Array map;
        var int mask, coefficient;
        var int cursorX, cursorY;
        var int newI,newJ;
        var int address;
        var int temp;

        if (c = String.newLine()){
            do Output.println();
            return;
        }
        if (c = String.backSpace()){
            do Output.backSpace();
            return;
        }

        let map=Output.getMap(c);
        let cursorX = (col/2);
        let cursorY = row*11;
        if((col&1)=0) {
            let coefficient = 1;
            let mask=255;
            let mask=~mask;
        } else {
            let coefficient=256;
            let mask=255;
        }

        while(k < 11){
            let address = 16384+(cursorY*32)+cursorX;
            let temp = Memory.peek(address);
            let temp =(temp&mask) + (map[k]*coefficient);
            do Memory.poke(address,temp);
            let cursorY = cursorY+1;
            let k = k+1;
        }

        if (col = 63){
            let newI = row+1;
            let newJ = 0;
        } else {
            let newI = row;
            let newJ = col+1;
        }
        if (newI = 23) {
            let newI = 0;
        }
```

```
264         do Output.moveCursor(newI, newJ);
265         return;
266     }
267 
268     /** displays the given string starting at the cursor location,
269      *  and advances the cursor appropriately. */
270     function void printString(String s) {
271         var int i, len;
272         let len = s.length();
273         while (i < len) {
274             do Output.printChar(s.charAt(i));
275             let i = i+1;
276         }
277         return;
278     }
279 
280     /** Displays the given integer starting at the cursor location,
281      *  and advances the cursor appropriately. */
282     function void printInt(int i) {
283         var String s;
284         let s = String.new(6);
285         do s.setInt(i);
286         do Output.printString(s);
287         return;
288     }
289 
290     /** Advances the cursor to the beginning of the next line. */
291     function void println() {
292         var int cursorX;
293         let cursorX = row+1;
294         if(cursorX = 23) {
295             let cursorX = 0;
296         }
297         do Output.moveCursor(cursorX, 0);
298         return;
299     }
300 
301     /** Moves the cursor one column back. */
302     function void backSpace() {
303         var int cursorX, cursorY;
304         if(col = 0){
305             let cursorX = row-1;
306             let cursorY = 63;
307         } else {
308             let cursorX = row;
309             let cursorY = col-1;
310         }
311         if (cursorX < 0) {
312             let cursorX = cursorX + 23;
313         }
314         do Output.moveCursor(cursorX, cursorY);
315         return;
316     }
317 }
```

8 Screen.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/Screen.jack
5
6   /**
7    * A library of functions for displaying graphics on the screen.
8    * The Hack physical screen consists of 512 rows (indexed 0..511, top to bottom)
9    * of 256 pixels each (indexed 0..255, left to right). The top left pixel on
10   * the screen is indexed (0,0).
11   */
12  class Screen {
13
14      static Array twoToThe;
15      static boolean currentColor;
16
17      /** Initializes the Screen. */
18      function void init() {
19          var int i;
20
21          do Screen.clearScreen();
22          do Screen.setColor(true);
23
24          let twoToThe=Array.new(16);
25          let twoToThe[0]=1;
26          let i = 0;
27          while(i < 15){
28              let twoToThe[i+1] = twoToThe[i] + twoToThe[i];
29              let i = i+1;
30          }
31          return;
32      }
33
34      /** Erases the entire screen. */
35      function void clearScreen() {
36          var int i;
37          while(i < 8192){
38              do Memory.poke(16384+i,0);
39              let i = i+1;
40          }
41          return;
42      }
43
44      /** Sets the current color, to be used for all subsequent drawXXX commands.
45       *  Black is represented by true, white by false. */
46      function void setColor(boolean b) {
47          let currentColor = b;
48          return;
49      }
50
51      /** Draws the (x,y) pixel, using the current color. */
52      function void drawPixel(int x, int y) {
53          var int address, bit;
54          var int mask;
55          var int value;
56
57          let address = 16384 + (y*32) + (x/16);
58          let bit = x & 15;
59          let mask = ~twoToThe[bit];
```

```
60          let value = Memory.peek(address);
61          let value = (value & mask) + (currentColor & twoToThe[bit]);
62          do Memory.poke(address,value);
63          return;
64      }
65
66      /** Draws a line from pixel (x1,y1) to pixel (x2,y2), using the current color. */
67      function void drawLine(int x1, int y1, int x2, int y2) {
68          var int dx, dy;
69          var int a, b;
70          var boolean reversed;
71          var int diff;
72
73          if(x1 > x2){
74              let a = x2;
75              let x2 = x1;
76              let x1 = a;
77
78              let b = y2;
79              let y2 = y1;
80              let y1 = b;
81          }
82          let dx = x2-x1;
83          let dy = y2-y1;
84
85          if(dx=0 | dy=0) {
86              do Screen.drawRectangle(Math.min(x1,x2),Math.min(y1,y2),Math.max(x1,x2),Math.max(y1,y2));
87              return;
88          }
89          if(dy < 0) {
90              let b = y2;
91              let y2 = y1;
92              let y1 = b;
93              let reversed = true;
94              let dy = -dy;
95          }
96          let a = 0;
97          let b = 0;
98
99          while(~(a>dx | b>dy)) {
100             if(reversed) {
101                 do Screen.drawPixel(x1+a,y2-b);
102             } else {
103                 do Screen.drawPixel(x1+a,y1+b);
104             }
105
106             if (diff < 0) {
107                 let a = a+1;
108                 let diff = diff + dy;
109             }
110             else{
111                 let b = b+1;
112                 let diff = diff - dx;
113             }
114         }
115         return;
116     }
117
118     /** Draws a filled rectangle whose top left corner is (x1, y1)
119      * and bottom right corner is (x2,y2), using the current color. */
120     function void drawRectangle(int x1, int y1, int x2, int y2) {
121         var int i,j;
122         let x2 = x2+1;
123         let y2 = y2+1;
124
125         let i = y1;
126         while(i<y2){
127             let j = x1;
```

```
128             while(j<x2){
129                 do Screen.drawPixel(j,i);
130                 let j = j+1;
131             }
132             let i = i+1;
133         }
134         return;
135     }
136 
137     /** Draws a filled circle of radius r<=181 around (x,y), using the current color. */
138     function void drawCircle(int x, int y, int r) {
139         var int dy;
140         var int sq;
141         // will overflow if r > 181
142         if(r > 181){
143             return;
144         }
145         let dy = -r;
146         while(~(dy>r)){
147             let sq=Math.sqrt((r*r)-(dy*dy));
148             do Screen.drawLine(x-sq,y+dy,x+sq,y+dy);
149             let dy=dy+1;
150         }
151 
152         return;
153     }
154 }
```

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 character strings. In addition for constructing and disposing
 * strings, the class features methods for getting and setting individual
 * characters of the string, for erasing the string's last character,
 * for appending a character to the string's end, and more typical
 * string-oriented operations.
 */
class String {
    field Array str;
    field int len;

    /** constructs a new empty string with a maximum length of maxLength
     *  and initial length of 0. */
    constructor String new(int maxLength) {
        if (maxLength < 0) {
            do Sys.error(14);
        } else {
            let str = Array.new(maxLength + 1);
        }
        let len = 0;
        return this;
    }

    /** Disposes this string. */
    method void dispose() {
        if (len > 0) {
            do str.dispose();
        }
        do Memory.deAlloc(this);
        return;
    }

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

    /** Returns the character at the j-th location of this string. */
    method char charAt(int j) {
        if (j<0 | j>len | j=len) {
            do Sys.error(15);
        }
        return str[j];
    }

    /** Sets the character at the j-th location of this string to c. */
    method void setCharAt(int j, char c) {
        if (j<0 | j>len | j=len) {
            do Sys.error(15);
        }
        let str[j]=c;
        return;
    }

```

```
60      /** Appends c to this string's end and returns this string. */
61      method String appendChar(char c) {
62          let str[len]=c;
63          let len=len+1;
64          return this;
65      }
66
67      /** Erases the last character from this string. */
68      method void eraseLastChar() {
69          if (len > 0) {
70              let len = len-1;
71          }
72          return;
73      }
74
75      /** Returns the integer value of this string,
76       *  until a non-digit character is detected. */
77      method int intValue() {
78          var int value, i;
79          let value = 0;
80
81          if (str[0] = 45){
82              let i = 1;
83          }
84          while(i < len){
85              let value = (value*10) + (str[i]-48);
86              let i = i+1;
87          }
88          if(str[0]=45){
89              let value = -value;
90          }
91          return value;
92      }
93
94      /** Sets this string to hold a representation of the given value. */
95      method void setInt(int val) {
96          var int d;
97          let len=0;
98          if(val<0){
99              do appendChar(45);
100             let val=-val;
101         }
102         let d= -val;
103         if(val=0){
104             do appendChar(48);
105             return;
106         }
107         if(val=d){
108             do appendChar(48+3);
109             let val=val-30000;
110         }
111         do setIntHelper(val);
112         return;
113     }
114
115     method void setIntHelper(int val){
116         var int q;
117         if(val<10){
118             do appendChar(48+val);
119             return;
120         }
121         let q=val/10;
122         do setIntHelper(q);
123         do appendChar(48+val-(q*10));
124         return;
125     }
126
127     /** Returns the new line character. */
```

```
128       function char newLine() {
129           return 128;
130       }
131
132       /** Returns the backspace character. */
133       function char backSpace() {
134           return 129;
135       }
136
137       /** Returns the double quote (") character. */
138       function char doubleQuote() {
139           return 34;
140       }
141   }
```

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 that supports various program execution 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 the program execution. */
    function void halt() {
        while(true) {}
        return;
    }

    /** Waits approximately duration milliseconds and returns.  */
    function void wait(int duration) {
        var int i, j;
        while(i < duration) {
            let j = 0;
            while(j < 150){
                let j = j+1;
            }
            let i = i+1;
        }
        return;
    }

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