

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

```
1 ***** TEST START *****
2
3     preparing sub.tar
4 dos2unix: converting file /tmp/bodek.A6A030/nand2tet/Project12/ransha/presubmission/testdir/stud/sub.tar/README to Unix form
5     checking sub.tar
6 testing ArrayTest
7 Test ArrayTest passed! Woohoo
8 testing MathTest
9 Test MathTest passed! Woohoo
10 testing MemoryTest
11 Test MemoryTest passed! Woohoo
12 Good luck on the manual tests :) See you on the 10/07!
13
14 ***** TEST END *****
```

2 README

```
1  nivkeren,ransha
2  =====
3  Niv Keren, ID 201478351, niv.keren@mail.huji.ac.il
4  Ran Shaham, ID 203781000, ran.shaham1@mail.huji.ac.il
5  =====
6
7          Project 12- Operating System
8          -----
9
10 Submitted Files
11 -----
12 README      - This file.
13 Array jack  - An Array object class
14 Keyboard.jack - Basic function for keyboard input.
15 Math.jack   - Basic math function
16 Memory.jack - Controll the computer memory
17 Output.jack - Printing text to the screen functions
18 Screen.jack - Drawing on the screen functions
19 String.jack - Basic string object.
20 Sys.jack    - Runs the operating system.
21
22
23 Run command
24 -----
25
26
27 Remarks
28 -----
29
30 * executed as suggested in the book design
31 * "Mr. President, if I've learned one thing today,
32   it's that sometimes you have to not give a ..."
33   Morty Smith [Rick and Morty]
```

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         if (~(size > 0)) {
14             do Sys.error(2);
15         }
16         return Memory.alloc(size);
17     }
18
19     /** De-allocates the array and frees its space. */
20     method void dispose() {
21         do Memory.deAlloc(this);
22         return;
23     }
24 }
```

4 Keyboard.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/Keyboard.jack
5
6  /**
7   * A library for handling user input from the keyboard.
8   */
9  class Keyboard {
10     static int KEYBOARD;
11     static int MAX_STRING;
12
13     /** Initializes the keyboard. */
14     function void init() {
15         let KEYBOARD = 24576;
16         let MAX_STRING = 80;
17         return;
18     }
19
20     /**
21      * Returns the ASCII code (as char) of the currently pressed key,
22      * or 0 if no key is currently pressed.
23      * Recognizes all ASCII characters, as well as the following extension
24      * of action keys:
25      * New line = 128 = String.newline()
26      * Backspace = 129 = String.backspace()
27      * Left Arrow = 130
28      * Up Arrow = 131
29      * Right Arrow = 132
30      * Down Arrow = 133
31      * Home = 134
32      * End = 135
33      * Page Up = 136
34      * Page Down = 137
35      * Insert = 138
36      * Delete = 139
37      * ESC = 140
38      * F1 - F12 = 141 - 152
39      */
40     function char keyPressed() {
41         var char key;
42         let key = Memory.peek(KEYBOARD);
43         return key;
44     }
45
46     /**
47      * Reads the next character from the keyboard.
48      * waits until a key is pressed and then released, then echoes
49      * the key to the screen, and returns the value of the pressed key.
50      */
51     function char readChar() {
52         var char firstKey, key;
53         let firstKey = 0;
54         while (firstKey = 0) {
55             let firstKey = Keyboard.keyPressed();
56         }
57         let key = firstKey;
58         while (key = firstKey) {
59             let key = Keyboard.keyPressed();
```

```

60     }
61     if ((firstKey > 31) & (firstKey < 129)) {
62         do Output.printChar(firstKey);
63     }
64     return firstKey;
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 char c;
73     var String s;
74
75     do Output.printString(message);
76     let s = String.new(MAX_STRING);
77     let c = 0;
78     while (~(c = String.newLine())) {
79         let c = Keyboard.readChar();
80         if (c = String.newLine()) {
81             return s;
82         }
83         if (c = String.backSpace()) {
84             if (~(s.length() = 0)) {
85                 do s.eraseLastChar();
86                 do Output.backSpace();
87             }
88         }
89         else {
90             do s.appendChar(c);
91         }
92     }
93     return s;
94 }
95
96 /**
97  * Prints the message on the screen, reads the next line
98  * (until a newline character) from the keyboard, and returns its
99  * integer value (until the first non numeric character).
100  */
101 function int readInt(String message) {
102     var String s;
103     let s = Keyboard.readLine(message);
104     return s.intValue();
105 }
106 }

```

5 Makefile

```
1  # --- Empty Makefile ---
2  SHELL=bash
3
4  all:
5      @echo "Done."
6
7  JACK_EXT=.jack
8  JACK_COMPILER=../tools/JackCompiler.sh
9  TAR_FILES=README Makefile *$(JACK_EXT)
10 TAR_FLAGS=-cvf
11 TAR_NAME=project12.tar
12 TAR=tar
13
14 tar:
15     $(TAR) $(TAR_FLAGS) $(TAR_NAME) $(TAR_FILES)
16
17 JACK_FILES=$(JACK_EXT)
18 TEST_DIR="Test/"
19 LINK=ln
20 links:
21     @echo Creating test files...
22     @for f in $(JACK_FILES); do \
23         f_clean=$(basename $$f $(JACK_EXT)); \
24         f_dir=$$f_clean$(TEST_DIR); \
25         $(LINK) $$f $$f_dir || break; \
26         echo Created link: $$f_dir$$f; \
27     done
28
29 compile:
30     @echo Compiling test directories...
31     @for d in *$(TEST_DIR); do \
32         $(JACK_COMPILER) $$d; \
33     done
34
35
36 clean:
37     @echo Removing test files...
38     @find . -mindepth 2 -maxdepth 2 -name "*$(JACK_EXT)" ! -name "Main.jack" -print -delete
39     @echo Removing vm files...
40     @find . -name "*.vm" -not -path "*/Tetris/*" -print -delete
41
42 .PHONY: all tar clean
```

6 Math.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/Math.jack
5
6  /**
7   * A basic math library.
8   */
9  class Math {
10
11     /** Initializes the library. */
12     function void init() {
13         return;
14     }
15
16     /** Returns the absolute value of x. */
17     function int abs(int x) {
18         if (x < 0) {
19             let x = -x;
20         }
21         return x;
22     }
23
24     function int shiftLeft(int x, int shift) {
25         while (shift > 0) {
26             let x = x + x;
27             let shift = shift - 1;
28         }
29         return x;
30     }
31
32     /** Returns the product of x and y. */
33     function int multiply(int x, int y) {
34         var int sum, shift, i, j;
35         // Special Cases
36         if (x = 1) {
37             return y;
38         }
39         else {
40             if (y = 1) {
41                 return x;
42             }
43             else {
44                 if (x = (-1)) {
45                     return (-y);
46                 }
47                 else {
48                     if (y = (-1)) {
49                         return (-x);
50                     }
51                 }
52             }
53         }
54         if ((x = 0) | (y = 0)) {
55             return 0;
56         }
57         // Powers of two:
58         if (y = 2) {
59             return x + x;
```



```

60     }
61     if (y = 4) {
62         return (x + x) + (x + x);
63     }
64     if (y = 8) {
65         return Math.shiftLeft(x, 3);
66     }
67     if (y = 16) {
68         return Math.shiftLeft(x, 4);
69     }
70     if (y = 32) {
71         return Math.shiftLeft(x, 5);
72     }
73     if (y = 64) {
74         return Math.shiftLeft(x, 6);
75     }
76     if (y = 128) {
77         return Math.shiftLeft(x, 7);
78     }
79     if (y = 256) {
80         return Math.shiftLeft(x, 8);
81     }
82     let sum = 0;
83     let shift = x;
84     let i = 0;
85     let j = 1;
86     while (i < 16) {
87         if (~(j & y = 0)) {
88             let sum = sum + shift;
89         }
90         let shift = shift + shift;
91         let j = j + j;
92         let i = i + 1;
93     }
94     return sum;
95 }
96
97 /** Returns the integer part of x/y. */
98 function int divide(int x, int y) {
99     var int q, qy;
100     // Check for division by zero
101     if (y = 0) {
102         do Sys.error(3);
103         return -1;
104     }
105     // Check for negative numbers
106     if ((x < 0) | (y < 0)) {
107         if (~(x < 0) & (y < 0)) {
108             // Only one is negative
109             return -Math.divide(Math.abs(x), Math.abs(y));
110         }
111         else {
112             // This means both are negative.
113             let x = Math.abs(x);
114             let y = Math.abs(y);
115         }
116     }
117     // Special Cases
118     if (y = 1) {
119         return x;
120     }
121     if (y > x) {
122         return 0;
123     }
124
125     if ((y + y) < 0) {
126         // Overflow..
127         let q = 0;

```

```

128     }
129     else {
130         let q = Math.divide(x, y + y);
131     }
132     let qy = Math.multiply(q, y);
133     if ((x - (qy + qy)) < y) {
134         return (q + q);
135     }
136     else {
137         return (q + q + 1);
138     }
139 }
140
141 /** Returns the integer part of the square root of x. */
142 function int sqrt(int x) {
143     var int y, j, two2j, z;
144     if (x < 0) {
145         do Sys.error(4);
146         return -1;
147     }
148     let y = 0;
149     let z = 0;
150     let j = 7;
151     let two2j = 128;
152     while (j > -1) {
153         if ((y + two2j) < 182) {
154             let z = y + two2j;
155             if (((z * z) - 1) < x) {
156                 let y = z;
157             }
158             if ((y * y) = x) {
159                 return y;
160             }
161         }
162         let two2j = two2j / 2;
163         let j = j - 1;
164     }
165     return y;
166 }
167
168 /** Returns the greater number. */
169 function int max(int a, int b) {
170     var int x;
171     let x = a;
172     if (b > a) {
173         let x = b;
174     }
175     return x;
176 }
177
178 /** Returns the smaller number. */
179 function int min(int a, int b) {
180     return -Math.max(-a, -b);
181 }
182 }

```

7 Memory.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/Memory.jack
5
6  /**
7   * Memory operations library.
8   */
9  class Memory {
10     static Array freeList, base;
11     static int len, next;
12
13     /** Initializes memory parameters. */
14     function void init() {
15         let base = 0;
16         let freeList = 2048;    // Heap base
17         let len = 0;
18         let next = 1;
19         let freeList[len] = 14336; // Heap size
20         let freeList[next] = null;
21         return;
22     }
23
24     /** Returns the value of the main memory at the given address. */
25     function int peek(int address) {
26         return base[address];
27     }
28
29     /** Sets the value of the main memory at this address
30      * to the given value. */
31     function void poke(int address, int value) {
32         let base[address] = value;
33         return;
34     }
35
36     /** Swaps two entries in the freeList data structure */
37     function void swap(Array curr, Array prev, Array prevprev) {
38         var Array tmpNext;
39
40         if (~(prevprev = null)) {
41             let prevprev[next] = curr;
42         }
43
44         let tmpNext = prev;
45         let prev[next] = curr[next];
46         let curr[next] = tmpNext;
47         return;
48     }
49
50     /** Sorts the freeList data structure */
51     function void sort() {
52         var Array curr, prev, prevprev, head;
53         var int i, n;
54         var boolean swapped;
55
56         if (freeList = null) {
57             return;
58         }
59     }
```

```

60     let swapped = true;
61     let prevprev = null;
62     let prev = freeList;
63     let curr = freeList[next];
64     let head = freeList;
65
66     let n = 1;
67     while (~(curr = null)) {
68         let prevprev = prev;
69         let prev = curr;
70         let curr = curr[next];
71         let n = n + 1;
72     }
73     while (swapped) {
74         let swapped = false;
75         let i = 1;
76         let prevprev = null;
77         let prev = head;
78         let curr = prev[next];
79
80         while (i < n) {
81             if (curr < prev) {
82                 if (i = 1) {
83                     let head = curr;
84                 }
85                 do Memory.swap(curr, prev, prevprev);
86                 let swapped = true;
87             }
88             let prevprev = prev;
89             let prev = curr;
90             let curr = curr[next];
91             let i = i + 1;
92         }
93         let n = n - 1;
94     }
95     let freeList = head;
96     return;
97 }
98
99 /** Performs defragmentation */
100 function void defrag() {
101     var Array prev, curr;
102
103     do Memory.sort();
104     let prev = freeList;
105     let curr = prev[next];
106     while (~(curr = null)) {
107         //do Memory.printList();
108         if ((prev + prev[len]) = curr) {
109             let prev[next] = curr[next];
110             let prev[len] = prev[len] + curr[len];
111             let curr = curr[next];
112         }
113         else {
114             let prev = curr;
115             let curr = curr[next];
116         }
117     }
118     return;
119 }
120
121 /** finds and allocates from the heap a memory block of the
122 * specified size and returns a reference to its base address. */
123 function int alloc(int size) {
124     var Array segment, prev, oldNext, output;
125     var int diff;
126
127     if (size < 1) {

```

```

128         do Sys.error(5);
129     }
130     if (freeList = null) {
131         do Sys.error(6); // Heap overflow
132     }
133
134     let prev = null;
135     let segment = freeList;
136
137     while ((size + 1) > segment[len]) {
138         let prev = segment;
139         let segment = segment[next];
140         if ((segment = null)) {
141             do Sys.error(6); // Even fragmentation didn't help!
142         }
143     }
144     // This is reached when a free block was found
145     let diff = segment[len] - size; // diff >= 1
146     // If the found segment is large enough to fit more than
147     // the required block, divide it to two blocks.
148     if (diff > 3) {
149         let output = segment + diff;
150         let output[-1] = size + 1;
151         let segment[len] = diff - 1;
152     }
153     else {
154         if (prev = null) {
155             let freeList = segment[next];
156         }
157         else {
158             let prev[next] = segment[next];
159         }
160         let output = segment + 1;
161     }
162     //do Memory.printList();
163     return output;
164 }
165
166 /** De-allocates the given object and frees its space. */
167 function void deAlloc(int object) {
168     var Array segment, curr, prev, nextSeg;
169
170     let curr = freeList;
171     let segment = object - 1;
172
173     if (freeList = null) {
174         let freeList = segment;
175         let freeList[next] = null;
176         return;
177     }
178
179     let nextSeg = freeList[next];
180
181     while (~(nextSeg = null)) {
182         let curr = nextSeg;
183         let nextSeg = nextSeg[next];
184     }
185     let curr[next] = segment;
186     let segment[next] = null;
187     do Memory.defrag();
188     return;
189 }
190 }

```

8 Output.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/Output.jack
5
6 /**
7  * Handles writing characters to the screen.
8  * The text screen (256 columns and 512 rows) is divided into 23 text rows (0..22),
9  * each containing 64 text columns (0..63).
10 * Each row is 11 pixels high (including 1 space pixel), and 8 pixels wide
11 * (including 2 space pixels).
12 */
13 class Output {
14
15     // Character map for printing on the left of a screen word
16     static Array charMaps;
17     static int row, col, MAX_ROW, MAX_COL, SCREEN;
18
19     /** Initializes the screen and locates the cursor at the screen's top-left. */
20     function void init() {
21         let row = 0;
22         let col = 0;
23         let MAX_ROW = 23;
24         let MAX_COL = 64;
25         let SCREEN = 16384;
26         do Output.initMap();
27         return;
28     }
29
30     // Initializes the character map array
31     function void initMap() {
32         var int i;
33
34         let charMaps = Array.new(127);
35
36         // black square (used for non printable characters)
37         do Output.create(0,63,63,63,63,63,63,63,63,63,0,0);
38
39         // Assigns the bitmap for each character in the character set.
40         do Output.create(32,0,0,0,0,0,0,0,0,0,0,0); //
41         do Output.create(33,12,30,30,30,12,12,0,12,12,0,0); // !
42         do Output.create(34,54,54,20,0,0,0,0,0,0,0,0); // "
43         do Output.create(35,0,18,18,63,18,18,63,18,18,0,0); // #
44         do Output.create(36,12,30,51,3,30,48,51,30,12,12,0); // £
45         do Output.create(37,0,0,35,51,24,12,6,51,49,0,0); // %
46         do Output.create(38,12,30,30,12,54,27,27,27,54,0,0); // &
47         do Output.create(39,12,12,6,0,0,0,0,0,0,0,0); // '
48         do Output.create(40,24,12,6,6,6,6,6,12,24,0,0); // (
49         do Output.create(41,6,12,24,24,24,24,24,12,6,0,0); // )
50         do Output.create(42,0,0,0,51,30,63,30,51,0,0,0); // *
51         do Output.create(43,0,0,0,12,12,63,12,12,0,0,0); // +
52         do Output.create(44,0,0,0,0,0,0,0,12,12,6,0); // ,
53         do Output.create(45,0,0,0,0,0,63,0,0,0,0,0); // -
54         do Output.create(46,0,0,0,0,0,0,0,12,12,0,0); // .
55         do Output.create(47,0,0,32,48,24,12,6,3,1,0,0); // /
56
57         do Output.create(48,12,30,51,51,51,51,51,30,12,0,0); // 0
58         do Output.create(49,12,14,15,12,12,12,12,12,63,0,0); // 1
59         do Output.create(50,30,51,48,24,12,6,3,51,63,0,0); // 2
```

```

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

```

```

128     do Output.create(115,0,0,0,30,51,6,24,51,30,0,0); // s
129     do Output.create(116,4,6,6,15,6,6,6,54,28,0,0); // t
130     do Output.create(117,0,0,0,27,27,27,27,27,54,0,0); // u
131     do Output.create(118,0,0,0,51,51,51,51,30,12,0,0); // v
132     do Output.create(119,0,0,0,51,51,51,51,63,63,18,0,0); // w
133     do Output.create(120,0,0,0,51,30,12,12,30,51,0,0); // x
134     do Output.create(121,0,0,0,51,51,51,62,48,24,15,0); // y
135     do Output.create(122,0,0,0,63,27,12,6,51,63,0,0); // z
136
137     do Output.create(123,56,12,12,12,7,12,12,12,56,0,0); // {
138     do Output.create(124,12,12,12,12,12,12,12,12,12,0,0); // |
139     do Output.create(125,7,12,12,12,56,12,12,12,7,0,0); // }
140     do Output.create(126,38,45,25,0,0,0,0,0,0,0,0); // ~
141
142     return;
143 }
144
145 // Creates a character map array of the given char index with the given values.
146 function void create(int index, int a, int b, int c, int d, int e,
147                     int f, int g, int h, int i, int j, int k) {
148     var Array map;
149
150     let map = Array.new(11);
151     let charMaps[index] = map;
152
153     let map[0] = a;
154     let map[1] = b;
155     let map[2] = c;
156     let map[3] = d;
157     let map[4] = e;
158     let map[5] = f;
159     let map[6] = g;
160     let map[7] = h;
161     let map[8] = i;
162     let map[9] = j;
163     let map[10] = k;
164
165     return;
166 }
167
168 // Returns the character map (array of size 11) for the given character
169 // If an invalid character is given, returns the character map of a black square.
170 function Array getMap(char c) {
171
172     if ((c < 32) | (c > 126)) {
173         let c = 0;
174     }
175
176     return charMaps[c];
177 }
178
179 /** Moves the cursor to the j'th column of the i'th row,
180  * and erases the character that was there. */
181 function void moveCursor(int i, int j) {
182     if ((i < 0) | (i > (MAX_ROW-1)) | (j < 0) | (j > (MAX_COL-1))) {
183         do Sys.error(20);
184         return;
185     }
186     let row = i;
187     let col = j;
188     do Output.drawChar(32);
189     return;
190 }
191
192 function void drawChar(char c) {
193     var Array map;
194     var int address, charRow, currWord, mask, shift;
195     var boolean firstInWord;

```



```

196
197     let map = Output.getMap(c);
198     let address = SCREEN + ((row * 352) + (col / 2)); // 32 * 11 = 352
199     if ((col & 1) = 0) {
200         let firstInWord = true;
201     }
202     else {
203         let firstInWord = false;
204     }
205
206     let mask = 255; // 0000000011111111
207     let shift = 256;
208     if (firstInWord) {
209         let mask = -256; // mask (shift 8)
210         let shift = 1;
211     }
212     let charRow = 0;
213     while (charRow < 11) {
214         let currWord = Memory.peek(address);
215         let currWord = (currWord & mask) | (map[charRow] * shift);
216         do Memory.poke(address, currWord);
217         let address = address + 32;
218         let charRow = charRow + 1;
219     }
220
221     return;
222 }
223
224
225 function void incCursor() {
226     let col = col + 1;
227     if (col > (MAX_COL - 1)) {
228         let col = 0;
229         let row = row + 1;
230     }
231     if (row > (MAX_ROW - 1)) {
232         let row = 0;
233     }
234     return;
235 }
236
237 /** Prints c at the cursor location and advances the cursor one
238  * column forward. */
239 function void printChar(char c) {
240     var Array map;
241     var int address, charRow, currWord, mask, shift, firstInWord;
242
243     // print new line
244     if (c = String.newLine()) {
245         do Output.println();
246         return;
247     }
248     // print backspace
249     if (c = String.backSpace()) {
250         do Output.backSpace();
251         return;
252     }
253
254     do Output.drawChar(c);
255
256     do Output.incCursor();
257     return;
258 }
259
260 /** Prints s starting at the cursor location, and advances the
261  * cursor appropriately. */
262 function void printString(String s) {
263     var int i, n;

```

```

264         let i = 0;
265         let n = s.length();
266         while (i < n) {
267             do Output.printChar(s.charAt(i));
268             let i = i + 1;
269         }
270         return;
271     }
272
273     /** Prints i starting at the cursor location, and advances the
274     * cursor appropriately. */
275     function void printInt(int i) {
276         var String s;
277
278         let s = String.new(10);
279         do s.setInt(i);
280         do Output.printString(s);
281
282         return;
283     }
284
285     /** Advances the cursor to the beginning of the next line. */
286     function void println() {
287         let col = 0;
288         let row = row + 1;
289         if (row > (MAX_ROW-1)) {
290             let row = 0;
291         }
292         return;
293     }
294
295     /** Moves the cursor one column back. */
296     function void backSpace() {
297         if (col = 0) {
298             return;
299         }
300         do Output.moveCursor(row, col-1);
301         return;
302     }
303 }

```

9 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   * Graphic screen library.
8   */
9  class Screen {
10     static boolean color;
11     static int SCREEN;
12     static Array bits;
13
14     /** Initializes the Screen. */
15     function void init() {
16         var int i, bit;
17         let color = true;
18         let SCREEN = 16384;
19
20         let bits = Array.new(16);
21         let i = 0;
22         let bit = 1;
23         while (i < 16) {
24             let bits[i] = bit;
25             let bit = bit + bit;
26             let i = i + 1;
27         }
28
29         return;
30     }
31
32     /** Erases the whole screen. */
33     function void clearScreen() {
34         var int i;
35         let i = 0;
36         while (i < 8192) {
37             do Memory.poke(SCREEN + i, 0);
38             let i = i + 1;
39         }
40         return;
41     }
42
43     /** Sets the color to be used in further draw commands
44      * where white = false, black = true. */
45     function void setColor(boolean b) {
46         let color = b;
47         return;
48     }
49
50     function boolean isValidPoint(int x, int y) {
51         if ((x < 0) | (y < 0) | (x > 511) | (y > 255)) {
52             return false;
53         }
54         return true;
55     }
56
57     function int div16(int x) {
58         var int result, i;
59         let i = 4;
```

```

60     let result = 0;
61     while (i < 16) {
62         if (~(bits[i] & x)=0)) {
63             let result = result + bits[i-4];
64         }
65         let i = i + 1;
66     }
67     return result;
68 }
69
70 /** Draws the (x, y) pixel. */
71 function void drawPixel(int x, int y) {
72     var int offset, xmod16, pixel, currWord;
73     if (~Screen.isValidPoint(x, y)) {
74         do Sys.error(7);
75         return; // This isn't reached
76     }
77     let offset = SCREEN + ((y * 32) + Screen.div16(x));
78     let xmod16 = x & 15;
79     let pixel = bits[xmod16];
80     let currWord = Memory.peek(offset);
81     if (color) {
82         let pixel = pixel | currWord;
83     }
84     else {
85         let pixel = (~pixel) & currWord;
86     }
87     do Memory.poke(offset, pixel);
88     return;
89 }
90
91 /** Draws a line from (x1, y1) to (x2, y2). */
92 function void drawLine(int x1, int y1, int x2, int y2) {
93     var int tmp, dx, dy, a, b, adyMinusbdx, inc, offset;
94
95     if ((~Screen.isValidPoint(x1,y1)) | (~Screen.isValidPoint(x2,y2))) {
96         do Sys.error(8);
97         return;
98     }
99
100     let inc = 1;
101     if (x1 = x2) {
102         if (y1 < y2) {
103             while (y1 < (y2 + 1)) {
104                 do Screen.drawPixel(x1, y1);
105                 let y1 = y1 + 1;
106             }
107             return;
108         }
109         else {
110             while (y2 < (y1 + 1)) {
111                 do Screen.drawPixel(x2, y2);
112                 let y2 = y2 + 1;
113             }
114             return;
115         }
116     }
117     if (y1 = y2) {
118         if (x2 < x1) {
119             let tmp = x2;
120             let x2 = x1;
121             let x1 = tmp;
122         }
123         while (x1 < (x2 + 1)) {
124             let offset = SCREEN + ((y1 * 32) + Screen.div16(x1));
125             if (((x1 & 15) = 0) & ((x1 + 15) < x2)) {
126                 do Memory.poke(offset, color);
127                 let x1 = x1 + 16;

```

```

128         }
129         else {
130             do Screen.drawPixel(x1, y1);
131             let x1 = x1 + 1;
132         }
133     }
134     return;
135 }
136 if (x2 < x1) {
137     let tmp = x1;
138     let x1 = x2;
139     let x2 = tmp;
140     let tmp = y1;
141     let y1 = y2;
142     let y2 = tmp;
143 }
144 if (y2 < y1) {
145     let inc = -1;
146 }
147
148 let dx = x2 - x1;
149 let dy = Math.abs(y2 - y1);
150
151 let a = 0;
152 let b = 0;
153 let adyMinusbdx = 0;
154
155 while (~(a > dx) | (b > dy)) {
156     do Screen.drawPixel(x1 + a, y1 + b);
157     if (adyMinusbdx < 0) {
158         let a = a + 1;
159         let adyMinusbdx = adyMinusbdx + dy;
160     }
161     else {
162         let b = b + inc;
163         let adyMinusbdx = adyMinusbdx - dx;
164     }
165 }
166 return;
167 }
168
169 /** Draws a filled rectangle where the top left corner
170 * is (x1, y1) and the bottom right corner is (x2, y2). */
171 function void drawRectangle(int x1, int y1, int x2, int y2) {
172     if ((~Screen.isValidPoint(x1,y1)) | (~Screen.isValidPoint(x2,y2)) |
173         (x1 > x2) | (y1 > y2)) {
174         do Sys.error(9);
175         return;
176     }
177     while (y1 < (y2 + 1)) {
178         do Screen.drawLine(x1,y1,x2,y1);
179         let y1 = y1 + 1;
180     }
181     return;
182 }
183
184 /** Draws a filled circle of radius r around (cx, cy). */
185 function void drawCircle(int cx, int cy, int r) {
186     var int dy, dx;
187     if (~Screen.isValidPoint(cx,cy)) {
188         do Sys.error(12);
189         return;
190     }
191     if ((~Screen.isValidPoint(cx + r,cy)) | (~Screen.isValidPoint(cx - r,cy)) |
192         (~Screen.isValidPoint(cx,cy + r)) | (~Screen.isValidPoint(cx,cy - r)) |
193         (r < 0)) {
194         do Sys.error(13);
195         return;

```

```
196     }
197     let dy = -r;
198     while (dy < (r + 1)) {
199         let dx = Math.sqrt((r*r) - (dy*dy));
200         do Screen.drawLine(cx + dx, cy + dy, cx - dx, cy + dy);
201         let dy = dy + 1;
202     }
203     return;
204 }
205 }
```

10 String.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/String.jack
5
6  /**
7   * Represents a String object. Implements the String type.
8   */
9  class String {
10     field Array charsArr;
11     field int size, maxSize;
12
13     /** Constructs a new empty String with a maximum length of maxLength. */
14     constructor String new(int maxLength) {
15         if (maxLength < 0) {
16             do Sys.error(14);
17             return this;
18         }
19         if (maxLength > 0) {
20             let charsArr = Array.new(maxLength);
21         }
22         let maxSize = maxLength;
23         let size = 0;
24         return this;
25     }
26
27     /** De-allocates the string and frees its space. */
28     method void dispose() {
29         if (maxSize > 0) {
30             do charsArr.dispose();
31         }
32         return;
33     }
34
35     /** Returns the current length of this String. */
36     method int length() {
37         return size;
38     }
39
40     /** Returns the character at location j. */
41     method char charAt(int j) {
42         if ((j < 0) | (j > (size-1))) {
43             do Sys.error(15);
44             return 0;
45         }
46         return charsArr[j];
47     }
48
49     /** Sets the j'th character of this string to be c. */
50     method void setCharAt(int j, char c) {
51         if ((j < 0) | (j > (maxSize-1))) {
52             do Sys.error(16);
53             return;
54         }
55         let charsArr[j] = c;
56         return;
57     }
58
59     /** Appends the character c to the end of this String.
```

```

60      * Returns this string as the return value. */
61  method String appendChar(char c) {
62      if (size = maxSize) {
63          do Sys.error(17);
64          return null;
65      }
66      do setCharAt(size, c);
67      let size = size + 1;
68      return this;
69  }
70
71  /** Erases the last character from this String. */
72  method void eraseLastChar() {
73      if (size = 0) {
74          do Sys.error(18);
75          return;
76      }
77      let size = size - 1;
78      return;
79  }
80
81  /** Returns the integer value of this String until the first non
82   * numeric character. */
83  method int intValue() {
84      var int numSize, i, result, minSize;
85      var boolean isNeg;
86      let isNeg = false;
87      let minSize = 0;
88      let numSize = 0;
89      let i = 1;
90      let result = 0;
91
92      if (size > 0) {
93          if (charsArr[0] = 45) {
94              let isNeg = true;
95              let numSize = numSize + 1;
96              let minSize = 1;
97          }
98      }
99
100     while ((charsArr[numSize] > 47) & (charsArr[numSize] < 58)) {
101         let numSize = numSize + 1;
102     }
103     while (numSize > minSize) {
104         let numSize = numSize - 1;
105         let result = result + ((charsArr[numSize] - 48) * i);
106         let i = i * 10;
107     }
108
109     if (isNeg) {
110         let result = -result;
111     }
112
113     return result;
114 }
115
116 /** Sets this String to hold a representation of the given number. */
117 method void setInt(int number) {
118     var int tmp, numlen;
119
120     let numlen = 1;
121     if (number = 0) {
122         if (maxSize = 0) {
123             do Sys.error(19);
124             return;
125         }
126         let charsArr[0] = 48;
127         return;

```



```

128     }
129     if (number < 0) {
130         let charsArr[0] = 45; // Add '-' sign
131         let numlen = numlen + 1;
132         let number = -number;
133     }
134     let tmp = number;
135     while ((tmp / 10) > 0) {
136         let numlen = numlen + 1;
137         let tmp = tmp / 10;
138     }
139     if (numlen > maxSize) {
140         do Sys.error(19);
141         return;
142     }
143     let size = numlen;
144     let tmp = 0;
145     while (number > 0) {
146         // str[lastDigit] = number % 10
147         let charsArr[numlen - tmp - 1] = (number - ((number / 10) * 10)) + 48;
148         let number = number / 10;
149         let tmp = tmp + 1;
150     }
151
152     return;
153 }
154
155 /** Returns the new line character. */
156 function char newLine() {
157     return 128;
158 }
159
160 /** Returns the backspace character. */
161 function char backSpace() {
162     return 129;
163 }
164
165 /** Returns the double quote (") character. */
166 function char doubleQuote() {
167     return 34;
168 }
169 }

```

11 Sys.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/Sys.jack
5
6  /**
7   * A library of basic system services.
8   */
9  class Sys {
10
11     /** Performs all the initializations required by the OS. */
12     function void init() {
13         do Memory.init();
14         do Math.init();
15         do Screen.init();
16         do Output.init();
17         do Keyboard.init();
18         do Main.main();
19         do Sys.halt();
20         return;
21     }
22
23     /** Halts execution. */
24     function void halt() {
25         while (true) {
26         }
27         return;
28     }
29
30     /** Waits approximately duration milliseconds and then returns. */
31     function void wait(int duration) {
32         var int i;
33
34         if (duration < 0) {
35             do Sys.error(1);
36         }
37         while (duration > 0) {
38             let i = 50;
39             while (i > 0) {
40                 let i = i - 1;
41             }
42             let duration = duration - 1;
43         }
44         return;
45     }
46
47     /** Prints the given error code in the form "ERR<errorCode>", and halts. */
48     function void error(int errorCode) {
49         do Output.printString("ERR");
50         do Output.printInt(errorCode);
51         do Sys.halt();
52         return;
53     }
54 }
```