Contents

1 Basic Test Results

```
********** TESTING FOLDER STRUCTURE START **********
Checking your submission for presence of invalid (non-ASCII) characters...
No invalid characters found.
Submission logins are: danasil,linorcohen
Is this OK?
**********  TESTING FOLDER STRUCTURE END  **********

********** PROJECT TEST START **********
Compiling your OS with the builtin JackCompiler.
Finished compiling your OS.
Testing ArrayTest.
ArrayTest passed.
Testing MemoryTest.
MemoryTest passed.
**********  PROJECT TEST END  **********

Note: the tests you see above are all the presubmission tests
for this project. The tests might not check all the different
parts of the project or all corner cases, so write your own
tests and use them!
```

2 AUTHORS

```
danasil,linorcohen
Partner 1: Dana Silutin, dana.silutin@mail.huji.ac.il, 209498724
Partner 2: Linor Cohen, linor.cohen@mail.huji.ac.il, 318861226
Remarks:
```

3 Array.jack

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

/**
 * Represents an array.
 * In the Jack language, arrays are instances of the Array class.
 * Once declared, the array entries can be accessed using the usual
 * syntax arr[i]. Each array entry can hold a primitive data type as
 * well as any object type. Different array entries can have different
 * data types.
 */
class Array {

    /** Constructs a new Array of the given size. */
    function Array new(int size) {
        return Memory.alloc(size);
    }

    /** Disposes this array. */
    method void dispose() {
         do Memory.deAlloc(this);
         return;
    }
}
```

4 Keyboard.jack

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).

/**
 * A library for handling user input from the keyboard.
 */
class Keyboard {
        static Array keyboard;
        /** Initializes the keyboard. */
        function void init() {
            let keyboard = 24576;
            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() {
            // Uses Memory.peek
            return Memory.peek(keyboard);
        }

        /**
         * 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() {
            // This should behave exactly like the built-in OS.
            // Pseudocode:
            // 1. display the cursor
            // 2. while (keyPressed() = 0): do nothing
            // 3. let c = code of the currently pressed key
            // 4. while (~(keyPressed() = 0)): do nothing
            // 5. display c at the current cursor location
            // 6. advance the cursor
            // 7. return c
            var char key;
            do Output.printChar("0");
            while(Keyboard.keyPressed() = 0){}
            let key = Keyboard.keyPressed();
            do Output.backSpace();
            if(key =  String.backSpace())
            {
                do Output.backSpace();
            }
            else{
                do Output.printChar(key);
            }

            while(~(Keyboard.keyPressed() = 0)){}
            return key;
        }

        /**
         * 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 if the current value
         * is longer than a single character.
         */
        function String readLine(String message) {
            // This should behave exactly like the built-in OS.
            // You can assume input is at most 64 characters long.
            // Why? Because this is the width of our screen!
            // Pseudocode:
            // 1. printString(message)
            // 2. let str = ""
            // 3. while true
            // 4.   let c = readChar()
            // 5.   if (c = newLine)
            // 6.      display newLine (if not displayed already by readChar())
            // 7.      return str
            // 8.   else if (c = backSpace)
            // 9.      remove the last character from str, if possible
            // 10.     move the cursor accordingly
            // 11.  else
            // 12.    str.appendChar(c)
            var String line;
            var char c;
            do Output.printString(message);
            let line = String.new(64);
            let c = Keyboard.readChar();
            while(~(c = String.newLine())){
                if(c = String.backSpace()){
                    do line.eraseLastChar();
                }
                else{
                    do line.appendChar(c);
                }
                let c = Keyboard.readChar();
            }
            do Output.println();
            return line;
        }

        /**
         * 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 integer value (until the first non-digit character in the
         * entered text is detected). Also handles user backspaces.
         */
        function int readInt(String message) {
            // This should behave exactly like the built-in OS.
            var String line;
            do Output.printString(message);
            let line =  Keyboard.readLine("");
            return line.intValue();
        }
}
```

5 Math.jack

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).

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

        static Array twoToThe;
        static int divisionSum;

        /** Initializes the library. */
        function void init() {
            var int i, twoPower;

            let twoToThe = Array.new(16);
            let twoPower = 1;
            let i = 0;
            while (i < 16){
                let twoToThe[i] = twoPower;
                let twoPower = twoPower + twoPower;
                let i = i + 1;
            }
            return;
        }

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

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

        /**
         * 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, shiftedx, i;
            var boolean iBit;

            // 1. let sum = 0
            let sum = 0;

            // 2. let shiftedx = x
            let shiftedx = x;

            let i = 0;
            // 3. for i = 0 ... n-1 do
            while (i < 16){
                // 4.   if ((i-th bit of y) == 1)
                let iBit = Math.bit(y,i);
                if (iBit){
                    // 5.      let sum = sum + shiftedx
                    let sum = sum + shiftedx;
                }
                  // 6.   let shiftedx = 2*shiftedx
                  let shiftedx = shiftedx + shiftedx;

                  let i = i + 1;
            }
            // 7. return sum
            return sum;
        }

        /**
         * Returns the integer part of x/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 divide(x,y) return the same value.
         */
        function int divide(int x, int y) {
            // This should be implemented without using multiplication.
            // Hint: updating the value of 2*q*y in the "if" statement requires
            // at most a single addition operator at every recursion step.

            var int xAbs, yAbs, sign, res;

            let divisionSum = 0;
            let xAbs = Math.abs(x);
            let yAbs = Math.abs(y);
            let sign = 1;
            if (((x < 0) & (y > 0)) | ((x > 0) & (y < 0))){
                let sign = -1;
            }
            let res = Math.divideHandler(xAbs, yAbs);
            return res*sign;
        }

        /** adds b to the sum division value */
        function void setDivisionSum(int b){
            let divisionSum = divisionSum + b;
            return;
        }

        /** recursive function for divide function */
        function int divideHandler(int x, int y){
            var int q;

            // 1. if if (y > x) or (y < 0) return 0 - handle overflow
            if ((y > x) | (y < 0)){
                return 0;
            }

            // 2. let q = divideHandler(x, 2*y)
            let q = Math.divideHandler(x, y + y);

            // 3. if (x - 2*q*y < y)
            if ((x - divisionSum) < y){
                // 4.   return 2*q
                do Math.setDivisionSum(0);
                return q + q;
            // 5. else
            } else {
                // 6.   return 2*q + 1
                do Math.setDivisionSum(y);
                return q + q + 1;
            }

        }

        /** gets square of a number without using multiplication. run time = O(log n) */
        function int getSquareNoMultiply(int n){
            var int x, res, p;

            if (n = 0){
                return 0;
            }

            let x = #n;

            if (n & 1){
                let res = ^x;
                let res = ^res;
                let res = res + 1;
                let p = Math.getSquareNoMultiply(x);
                let p = ^p;
                let p = ^p;
                return p + res;
            } else{
                let p = Math.getSquareNoMultiply(x);
                let p = ^p;
                let p = ^p;
                return p;
            }
        }


        /** Returns the integer part of the square root of x. */
        function int sqrt(int x) {
            // This should be implemented without using multiplication or division.
            var int y, j, res, twoPowerJ;

            if (~(x > 0)) { // handle x = 0 or x < 0
                return 0;
            }

            // 1. let y = 0
            let y = 0;
            let j = 7;

            // 2. for j = (n/2 - 1) ... 0 do
            while (~(j < 0)){
                // 3.   if ((y + 2**j)**2 <= x) & ((y + 2**j)**2 > 0) then let y = y + 2**j
                let twoPowerJ = twoToThe[j];
                let res = Math.getSquareNoMultiply(y+twoPowerJ);
                if (~(res > x) & (res > 0)){
                    let y = y + twoPowerJ;
                }
                let j = j - 1;
            }
            // 4. return y
            return y;
        }

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

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

6 Memory.jack

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).
// 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 freeList;
    static Array prev;
    static int length;
    static int next;

    /** Initializes the class. */
    function void init() {
        let ram = 0;
        // 1. freeList = heapBase
        let freeList = 2048;

        // 2. freeList.length = heapLength
        let length = 0;
        let freeList[length] = 14335;

        // 3. freeList.next = null
        let next = 1;
        let freeList[next] = null;

        let prev = 0;
        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. Assume size > 0. */
    function int alloc(int size) {
        var Array foundBlock, nextAddress;
        var int prevLength;

        // 1. Search freeList using best-fit or first-fit heuristics to obtain
        //    a segment with segment.length >= size + 2.
        //    If no such segment is found, return -1.
        let foundBlock = Memory.firstFit(size);
        if (foundBlock = -1){
            return -1;
        }

        // 2. block = needed part of the found segment (or all of it, if the
        //            segment remainder is too small).
        let prevLength = foundBlock[length];

        // 3. block[-1] = size + 1 // Remember block size, for de-allocation
        let foundBlock[length] = size;
        let foundBlock[next] = null;

        // 4. Update freeList to reflect the allocation
        let nextAddress = foundBlock + size + 2;
        let nextAddress[next] = foundBlock[next];
        let nextAddress[length] = prevLength - size - 2;
        if (prev = 0){ // no elements in llist
            let freeList = nextAddress;
        }else{
            let prev[next] = nextAddress;
        }

        // 5. Return block
        return foundBlock + 2;

        // The figure MemoryTest/MemoryFreeListExample.png illustrates how
        // the freeList should look like.
    }

    /** first fit algorithm */
    function Array firstFit(int size){
        var Array cur;

        let cur = freeList;
        let prev = 0;
        while (cur[length] < size + 2){
            let prev = cur;
            let cur = cur[next];
            if (cur = null){
                return -1;
            }
        }
        return cur;
    }

    /** De-allocates the given object (cast as an array) by making
     *  it available for future allocations. */
    function void deAlloc(Array o) {
        var Array segment;
        // 1. segment = 0 - 2
        let segment = o - 2;
        // add at the beginning of the free list
        let segment[next] = freeList;
        let freeList = segment;
        return;
    }

    /** Returns the maximal element contained in the given Array/object.
     *  Assume inputs are initialized and contain at least one element. */
    function int max(Array o) {
        // Hint: the algorithm you need to implement in Memory.alloc saves the
        // size of the allocated block in the memory cell right before the
        // start of the block, and this can be used to implement Memory.max.

        var int maxVal, size, i, val;

        let size = Memory.peek(o-2);
        let maxVal = 0;
        let i = 0;
        while (i < size){
            let val = Memory.peek(o + i);
            if (val > maxVal){
                let maxVal = val;
            }
            let i = i + 1;
        }
        return maxVal;
    }
}
```

7 Output.jack

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).
// 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;
    static int col;
    static Array screen;

    /** Initializes the screen, and locates the cursor at the screen's top-left. */
    function void init() {
        do Output.initMap();
        let row = 0;
        let col = 0;
        let screen = 16384;
        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
        do Output.create(49,12,14,15,12,12,12,12,12,63,0,0); // 1
        do Output.create(50,30,51,48,24,12,6,3,51,63,0,0);   // 2
        do Output.create(51,30,51,48,48,28,48,48,51,30,0,0); // 3
        do Output.create(52,16,24,28,26,25,63,24,24,60,0,0); // 4
        do Output.create(53,63,3,3,31,48,48,48,51,30,0,0);   // 5
        do Output.create(54,28,6,3,3,31,51,51,51,30,0,0);    // 6
        do Output.create(55,63,49,48,48,24,12,12,12,12,0,0); // 7
        do Output.create(56,30,51,51,51,30,51,51,51,30,0,0); // 8
        do Output.create(57,30,51,51,51,62,48,48,24,14,0,0); // 9

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

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

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

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

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

    return;
    }

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

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

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

        return;
    }

    // Returns the character map (array of size 11) of the given character.
    // If the given character is invalid or non-printable, returns the
    // character map of a black square.
    function Array getMap(char c) {
        if ((c < 32) | (c > 126)) {
            let c = 0;
        }
        return charMaps[c];
    }

    /** Moves the cursor to the j-th column of the i-th row,
     *  and erases the character displayed there. */
    function void moveCursor(int i, int j) {
        let row = i;
        let col = j;
        do Output.printChar(32); // erase the character that was there
        let row = i;
        let col = j;
        return;
    }


    /** Displays the given character at the cursor location,
     *  and advances the cursor one column forward. */
    function void printChar(char c) {
        // Your implementation should be as efficient as possible.
        // For example, you can draw multiple pixels at once, for each row.
        var Array c;
        var int address, mask, i;
        let c = Output.getMap(c);
        let address = (col/2) + (22*16*row);
        let mask = col&1;
        let i = 0;
        while(i < 11)
        {
            if(mask = 0)
            {
                let screen[address] = (screen[address] & -256) + c[i];
            }
            else
            {
                let screen[address] = (screen[address] & 255) + (c[i]*256);
            }
            let address = address + 32;
            let i = i+1;
        }
        if(col = 63)
        {
            do Output.println();
        }
        else{
            let col = col + 1;
        }
        return;
    }


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

        return;
    }

    /** Displays the given integer starting at the cursor location,
     *  and advances the cursor appropriately. */
    function void printInt(int i) {
        var String num;
        let num = String.new(16);
        do num.setInt(i);
        do Output.printString(num);
        do num.dispose();
        return;
    }

    /** Advances the cursor to the beginning of the next line. */
    function void println() {
        if(row < 22){
            do Output.moveCursor((row+1),0);
        }
        else{
            do Output.moveCursor(0,0);
        }
        return;
    }

    /** Moves the cursor one column back. */
    function void backSpace() {
        if (col = 0){
            if(row=0){
                do Output.moveCursor(0, 0);
            }
            do Output.moveCursor((row - 1), 63);
        }
        else {
            do Output.moveCursor(row, (col - 1));
        }
        return;
        }
}

```

8 Screen.jack

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).

/**
 * A library of functions for displaying graphics on the screen.
 * The Hack physical screen consists of 256 rows (indexed 0..255, top to bottom)
 * of 512 pixels each (indexed 0..511, left to right). The top left pixel on
 * the screen is indexed (0,0).
 */
class Screen {
        static Array screen;
        static boolean color;
        static Array helper;

        /** Initializes the Screen. */
        function void init() {
            var int i;
            let screen = 16384; //set first address;
            let color = true;
            let i= 1;
            let helper = Array.new(16);
            let helper[0] = 1;
            while (i < 16) {
                let helper[i] = helper[i-1] + helper[i-1];
                let i = i + 1;
            }
            return;
        }

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

        /** Sets the current color, to be used for all subsequent drawXXX commands.
         *    Black is represented by true, white by false. */
        function void setColor(boolean b) {
            let color = b;
            return;
        }

        /** Draws the (x,y) pixel, using the current color. */
        function void drawPixel(int x, int y) {
            // For this function, you will need to calculate the value x%16.
            // It should be calculated quickly without using multiplication or
            // division, using exactly one basic math/logical operation.
            // In addition, calculating 16384 + y * 32 + x/16 should not use
            // division or multiplication.
            // Pseudocode:
            // 1. Compute the RAM address where the (x,y) pixel is
            //    represented: 16384 + (32*y) + (x/16).
            // 2. Use Memory.peek to get the 16-bit value of this address
            // 3. Use some bitwise operation to set (only) the bit that corresponds
            //    to the pixel to the current color.
            // 4. Use Memory.poke to write the modified 16-bit value to the RAM
            // address.
            var int i, address, value, shiftx, shifty;
            let i=0;
            let shifty = Screen.mult32(y);
            let shiftx = Screen.div16(x);
            let address  = 16384 + shifty + shiftx;
            let value = Memory.peek(address);
            if(color){
                let value = value | helper[x & 15];
            }
            else{
                let value = value & ~ helper[x & 15];
            }
            do Memory.poke(address, value);
            return;
        }

        /** Draws a line from pixel (x1,y1) to pixel (x2,y2), using the current color. */
        function void drawLine(int x1, int y1, int x2, int y2) {
            // The case where x1 != x2 and y1 != y2 should be implemented without
            // multiplication or division.
            // Pseudocode:
            // 1. let x = x1, y = y1, a = 0, b = 0, diff = 0
            // 2. Compute dx and dy
            // 3. while ((a <= dx) and (b <= dy))
            // 4.   do drawPixel(x+a, y+b)
            // 5.   if (diff < 0) { let a=a+1, diff=diff+dy }
            // 6.   else          { let b=b+1, diff=diff-dx }
            //
            // The case where y1 == y2 can be implemented efficiently by
            // trying to draw as many pixels as possible in a single assignment,
            // similarly to Fill in project 4.
            var int x,y,a,b,diff,dx,dy,shifty,shiftx,i, temp, address;
            if(x1 > x2){
                let temp = x1;
                let x1 = x2;
                let x2 = temp;

                let temp = y1;
                let y1 = y2;
                let y2 = temp;
                }
            let dy = y1 - y2;
            let y=y1;
            let dx = x2 - x1;
            let x=x1;
            let a=0;
            let b=0;
            let diff = 0;
            let i=0;
            if((~(dx= 0)) & (~(dy = 0))){
                while((a<(dx+1)) & (b<(dy+1)))
                {
                    do Screen.drawPixel(x+a, y-b);
                    if(diff <0)
                    {
                        let a=a+1;
                        let diff = diff+dy;
                    }
                    else{
                        let b = b+1;
                        let diff = diff-dx;
                    }
                }

                while((a<(dx+1)) & (b>dy)){
                    do Screen.drawPixel(x+a, y+b);
                        if(diff <0)
                        {
                            let a=a+1;
                            let diff = diff-dy;
                        }
                        else{
                            let b = b+1;
                            let diff = diff-dx;
                        }
                    }
            }
            else{
                // draw horizontal line
                if(~(dx=0)){
                    let temp = x+a;
                    //draw pixels until x+a%16=0
                    while(((temp&15) > 0) & (a<(dx+1)) ){
                         do Screen.drawPixel(temp, y);
                         let a = a+1;
                         let temp = x+a;
                    }
                    //assign 16-bit values until less then 16 pixels are left
                    while((dx-a)>16){
                        let shifty = Screen.mult32(y);
                        let shiftx = Screen.div16(x+a);
                        let address = screen + shifty + shiftx;
                        if(color){
                            do Memory.poke(address, -1);
                        }
                        else{
                            do Memory.poke(address, 0);
                        }
                            let a = a + 16;
                    }
                    //draw last pixels
                    while(a<(dx+1)){
                         do Screen.drawPixel((x+a), y);
                         let a = a+1;
                    }
                }
                else
                {
                    if(dy < 0){
                        let dy = -dy;
                        let y = y2;
                    }
                    //draw vertical line
                    while(b<(dy+1)){
                    do Screen.drawPixel(x, (y-b));
                        let b=b+1;
                        let diff = diff+dy;
                    }
                }
            }

            return;

        }


        function int div16(int x){
            var int i;
            let i =0;
            while(i<4)
```

```
            {
                let x = #x;
                let i = i + 1;
            }
            return x;
        }

        function int mult32(int x){
            var int i;
            let i=0;
            while(i<5)
            {
                let x = ^x;
                let i = i + 1;
            }
            return x;
        }

        /** Draws a filled rectangle whose top left corner is (x1, y1)
         * and bottom right corner is (x2,y2), using the current color. */
        function void drawRectangle(int x1, int y1, int x2, int y2) {
        var int dx, dy, i, cury;
        let dx = x2 - x1;
        let dy = y2 - y1;
        let i = 0;
        //change to vertical lines
        while (i < (dy+1)) {
            let cury = y1+i;
            do Screen.drawLine(x1,cury,x2,cury);
            let i = i+1;
        }
        return;
        }

        /** Draws a filled circle of radius r<=181 around (x,y), using the current color. */
        function void drawCircle(int x, int y, int r) {
            // This can be implemented efficiently by drawing multiple lines at
            // each iteration. You can use multiplication and sqrt.
            // Pseudocode:
            // 1. for (dy = -r ... r)
            // 2.   let halfWidth = sqrt(r*r - dy*dy)
            // 3.   do drawLine(x-halfWidth, y+dy, x+halfWidth, y+dy)
            var int dx, dy;
            let dy = -r;
            while (dy < r) {
                let dx = Math.sqrt((r * r) - (dy * dy));
                do Screen.drawRectangle(x - dx, y + dy, x + dx, y + dy);
                do Screen.drawRectangle(x - dx, y - dy, x + dx, y - dy);
                let dy = dy + 1;
            }
            return;
        }
}
```

9 String.jack

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).

/**
 * 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 int maxLen;
    field int curLength;
    field Array string;

    /** constructs a new empty string with a maximum length of maxLen
     *  and initial length of 0. */
    constructor String new(int maxLength) {
        let curLength = 0;
        if (maxLength > 0){
            let string = Array.new(maxLength);
            let maxLen = maxLength;
            return this;
        }
        let maxLen = 1;
        let string = Array.new(maxLen);
        return this;
    }

    /** Disposes this string. */
    method void dispose() {
        do string.dispose();
        return;
    }

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

    /** Returns the character at the j-th location of this string. */
    method char charAt(int j) {
        return string[j];
    }

    /** Sets the character at the j-th location of this string to c. */
    method void setCharAt(int j, char c) {
        let string[j] = c;
        return;
    }

    /** Appends c to this string's end and returns this string. */
    method String appendChar(char c) {
        if (curLength < maxLen){
            let string[curLength] = c;
            let curLength = curLength + 1;
        }
        return this;
    }

    /** Erases the last character from this string. */
    method void eraseLastChar() {
        if ((curLength < 0) | (curLength = 0) ){
            return;
        }
        let curLength = curLength - 1;
        return;
    }

    /** Returns the integer value of this string,
     *  until a non-digit character is detected. */
    method int intValue() {
        var int val, i, d, sign;
        var int c;

        // handle negative value
        if ((curLength > 0) & (string[0] = 45)){
            let sign = -1;
        }else{
            let sign = 1;
        }

        // 1. let val = 0
        let val = 0;

        // 2. for (i = 0 .. str.length()) do
        if (sign = 1){
            let i = 0;
        } else {
            let i = 1;
        }
        while ((i < curLength) & ((string[i] > 47) & (string[i] < 58))){
                // 3.   let d = integer value of str.charAt(i)
                let c = string[i];
                let d = c - 48;

                // 4.   let val = (val*10) + d
                let val = (val*10) + d;

                let i = i + 1;
        }
        // 5. return val
        return val*sign;
    }

    /** Sets this string to hold a representation of the given value. */
    method void setInt(int val) {

        let curLength = 0;

        // handle negative value
        if (val < 0){
            do appendChar(45);
            let val = 0 - val;
        }
        do set2Int(val);
        return;
    }

    /** convert int to char */
    method char castIntToChar(int num) {
        return num;
    }

    /** set2Int recursively , so no need to clear the string each time */
    method void set2Int(int val){
            var int lastDigit, subRes;
            var char c;

            // 1. let lastDigit = val % 10
            let subRes = val / 10;
            let subRes = 10*subRes;
            let lastDigit = val - subRes;

            // 2. let c = character representing lastDigit
            let c = castIntToChar(lastDigit + 48);

            // 3. if (val < 10)
            if (val < 10){

                // 4.   do c (as a string)
                do appendChar(c);

            // 5. else
            } else{

                // 6.   do int2String(val / 10).appendChar(c)
                do set2Int(val / 10);
                do appendChar(c);
            }
            return;
    }

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

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

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

10 Sys.jack

```
// This file is part of nand2tetris, as taught in The Hebrew University, and
// was written by Aviv Yaish. It is an extension to the specifications given
// [here](https://www.nand2tetris.org) (Shimon Schocken and Noam Nisan, 2017),
// as allowed by the Creative Common Attribution-NonCommercial-ShareAlike 3.0
// Unported [License](https://creativecommons.org/licenses/by-nc-sa/3.0/).

/**
 * A library that supports various program execution services.
 */
class Sys {
    /** Performs all the initializations required by the OS. */
    function void init() {
        // Pseudocode:
        // 1. for each relevant OS class, do Class.init
        //    Some OS classes depend on others, so order is important here!
        // 2. do Main.main()
        // 3. do Sys.halt()
        do Memory.init();
        do Math.init();
        do Keyboard.init();
        do Output.init();
        do Screen.init();
        do Main.main();
        do Sys.halt();
        return;
    }

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

    /** Waits approximately duration milliseconds and returns.  */
    function void wait(int duration) {
        var int i, j;
        let i=0;
        while(i<duration){
            let j=100;
            while(j>0){
                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.printString(errorCode);
        do Sys.halt();
        return;
    }
}
```