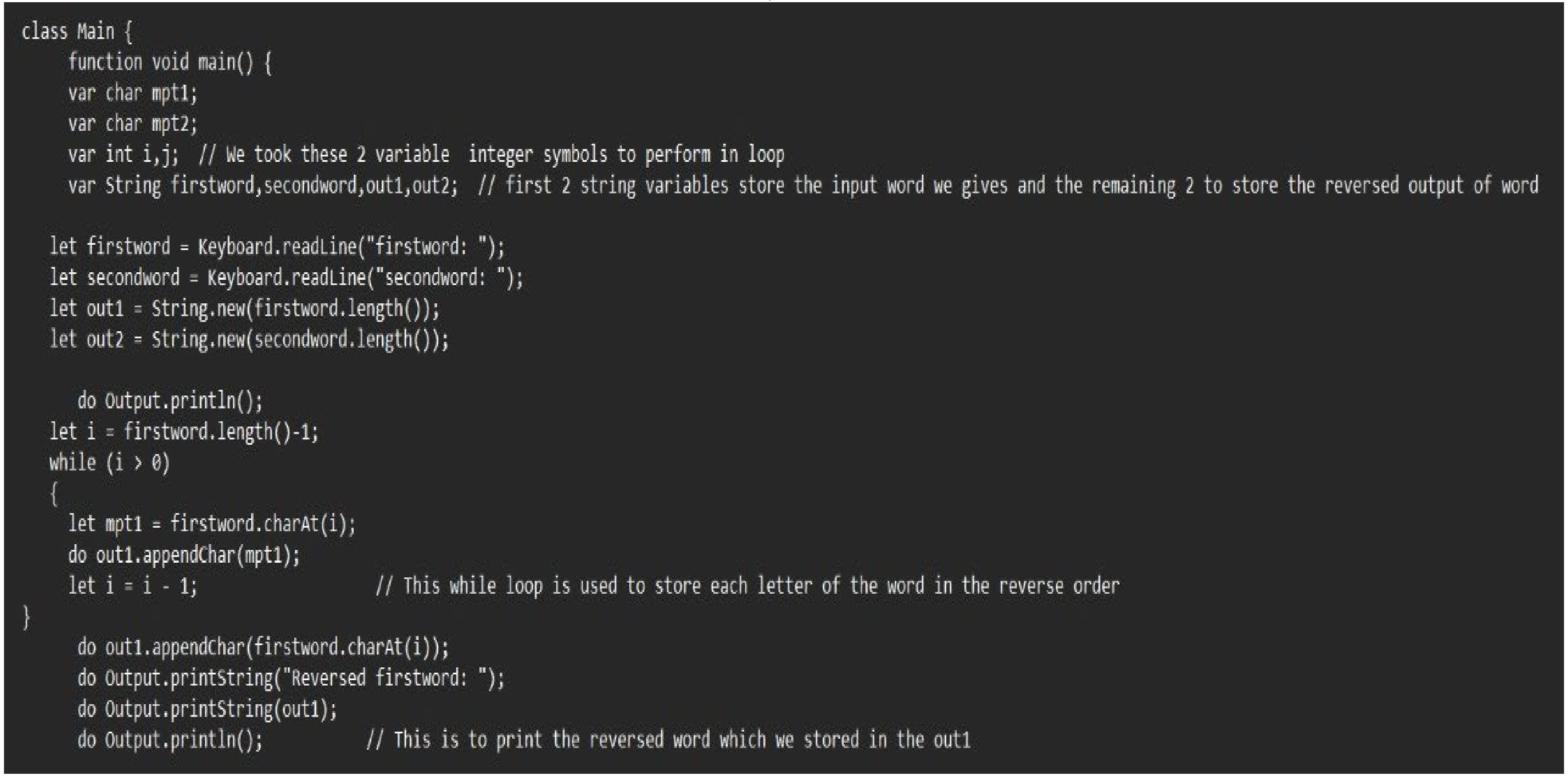
ELEMENTS OF COMPUTING

1

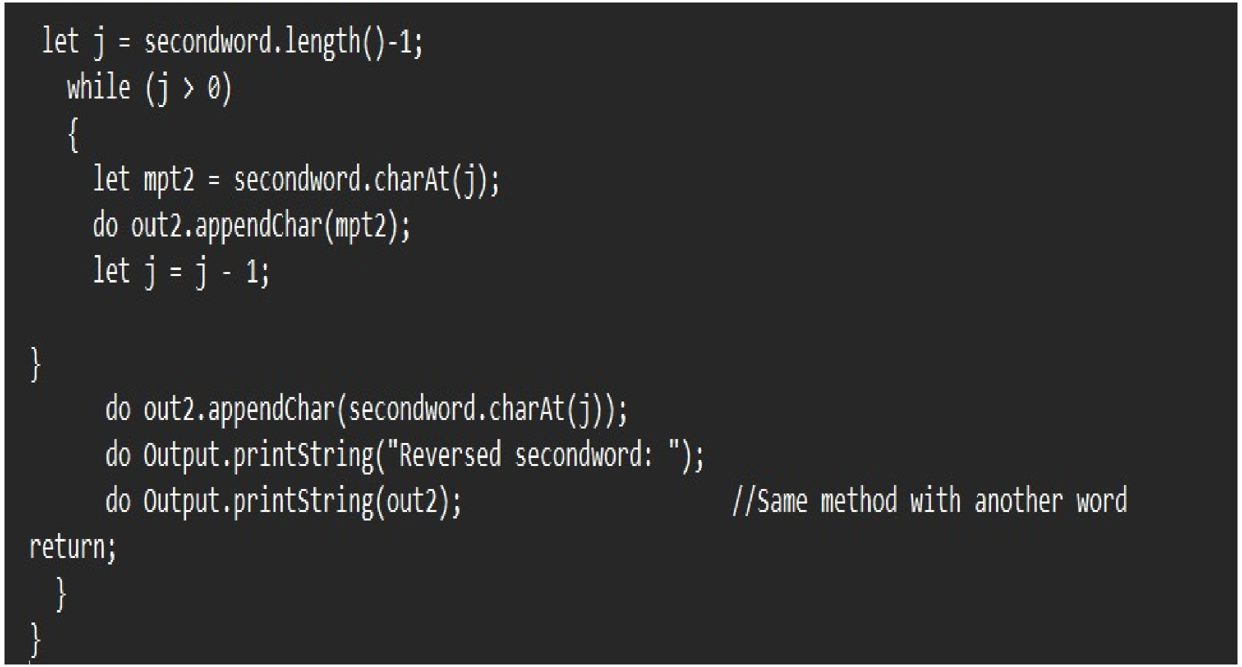
Write a Jack code to reverse the strings “Avengers” and “redivider”.

CODE



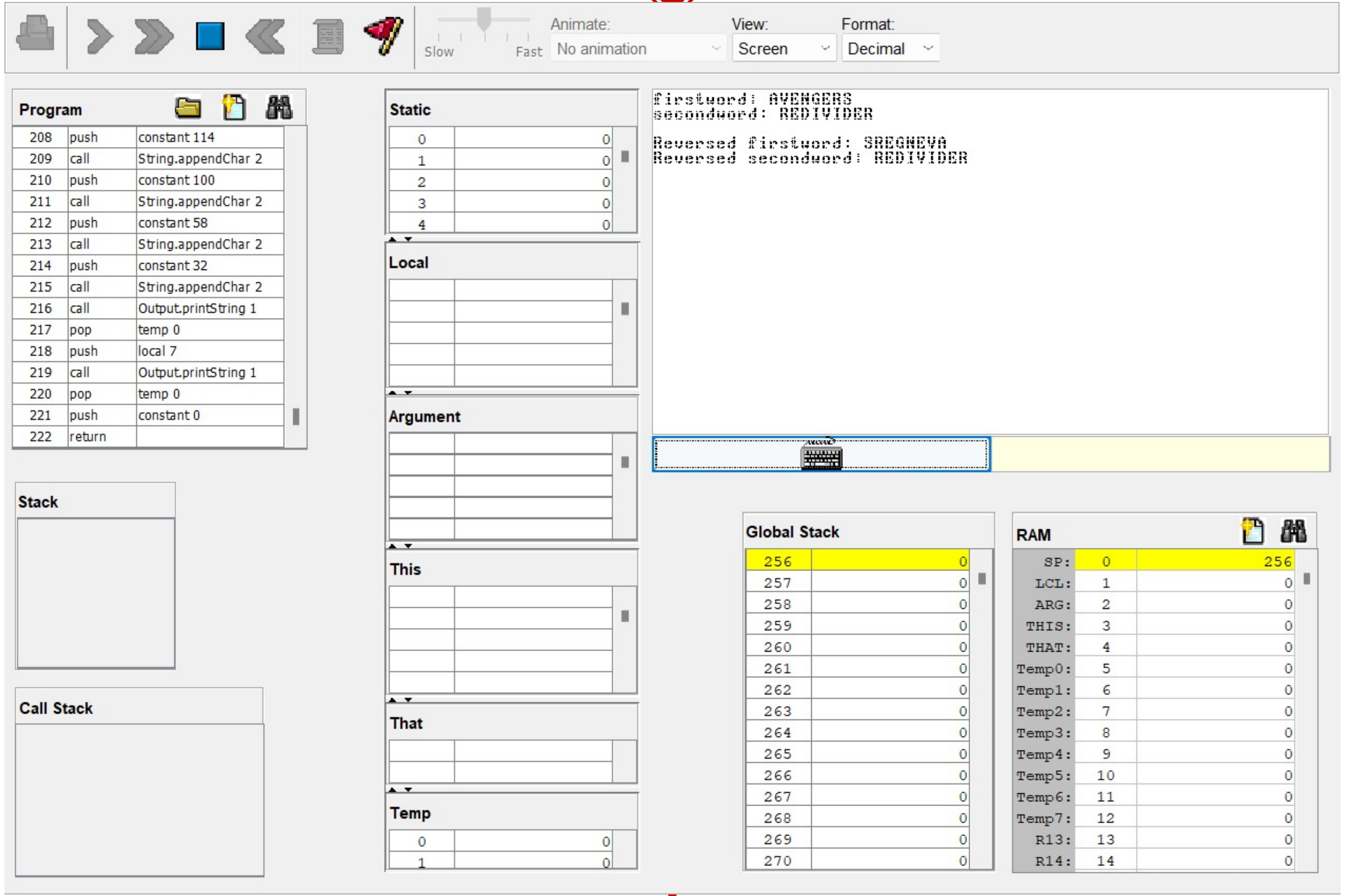
```
class Main {
     function void main() {
     var char mpt1;
     var char mpt2;
     var int i,j;  // We took these 2 variable  integer symbols to perform in loop
     var String firstword,secondword,out1,out2;  // first 2 string variables store the input word we gives and the remaining 2 to store the reversed output of word

   let firstword = Keyboard.readLine("firstword: ");
   let secondword = Keyboard.readLine("secondword: ");
   let out1 = String.new(firstword.length());
   let out2 = String.new(secondword.length());

      do Output.println();
   let i = firstword.length()-1;
   while (i > 0)
   {
     let mpt1 = firstword.charAt(i);
     do out1.appendChar(mpt1);
     let i = i - 1;                   // This while loop is used to store each letter of the word in the reverse order
}
      do out1.appendChar(firstword.charAt(i));
      do Output.printString("Reversed firstword: ");
      do Output.printString(out1);
      do Output.println();          // This is to print the reversed word which we stored in the out1
```

CODE



```
 let j = secondword.length()-1;
   while (j > 0)
   {
     let mpt2 = secondword.charAt(j);
     do out2.appendChar(mpt2);
     let j = j - 1;

}
      do out2.appendChar(secondword.charAt(j));
      do Output.printString("Reversed secondword: ");
      do Output.printString(out2);                     //Same method with another word
return;
  }
}
```

OUTPUT



Animate: No animation
View: Screen
Format: Decimal

```
firstword: AVENGERS
secondword: REDIVIDER

Reversed firstword: SREGNEVA
Reversed secondword: REDIVIDER
```

Program

208	push	constant 114
209	call	String.appendChar 2
210	push	constant 100
211	call	String.appendChar 2
212	push	constant 58
213	call	String.appendChar 2
214	push	constant 32
215	call	String.appendChar 2
216	call	Output.printString 1
217	pop	temp 0
218	push	local 7
219	call	Output.printString 1
220	pop	temp 0
221	push	constant 0
222	return	

Static

0	0
1	0
2	0
3	0
4	0

Temp

0	0
1	0

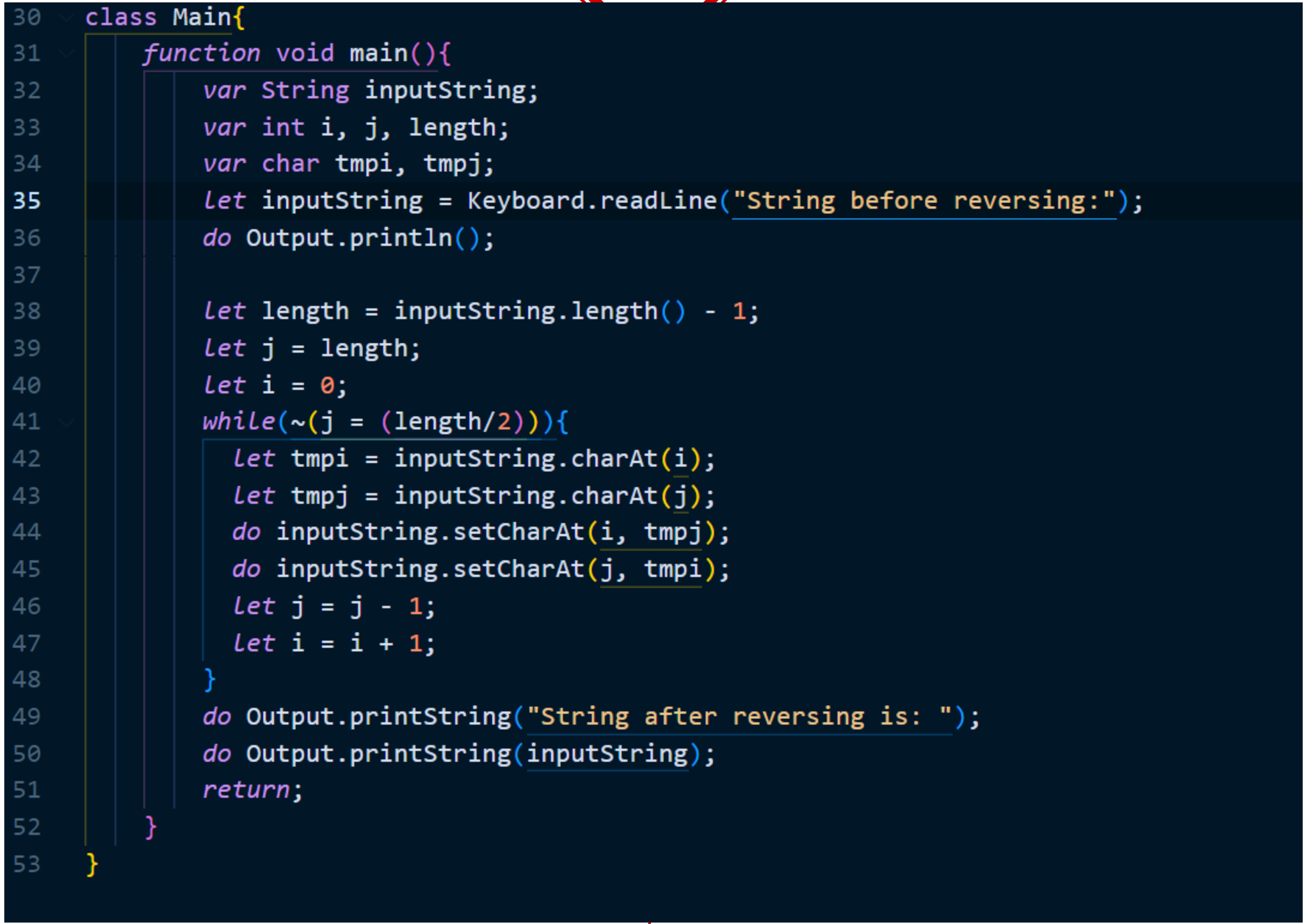
Global Stack

256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

RAM

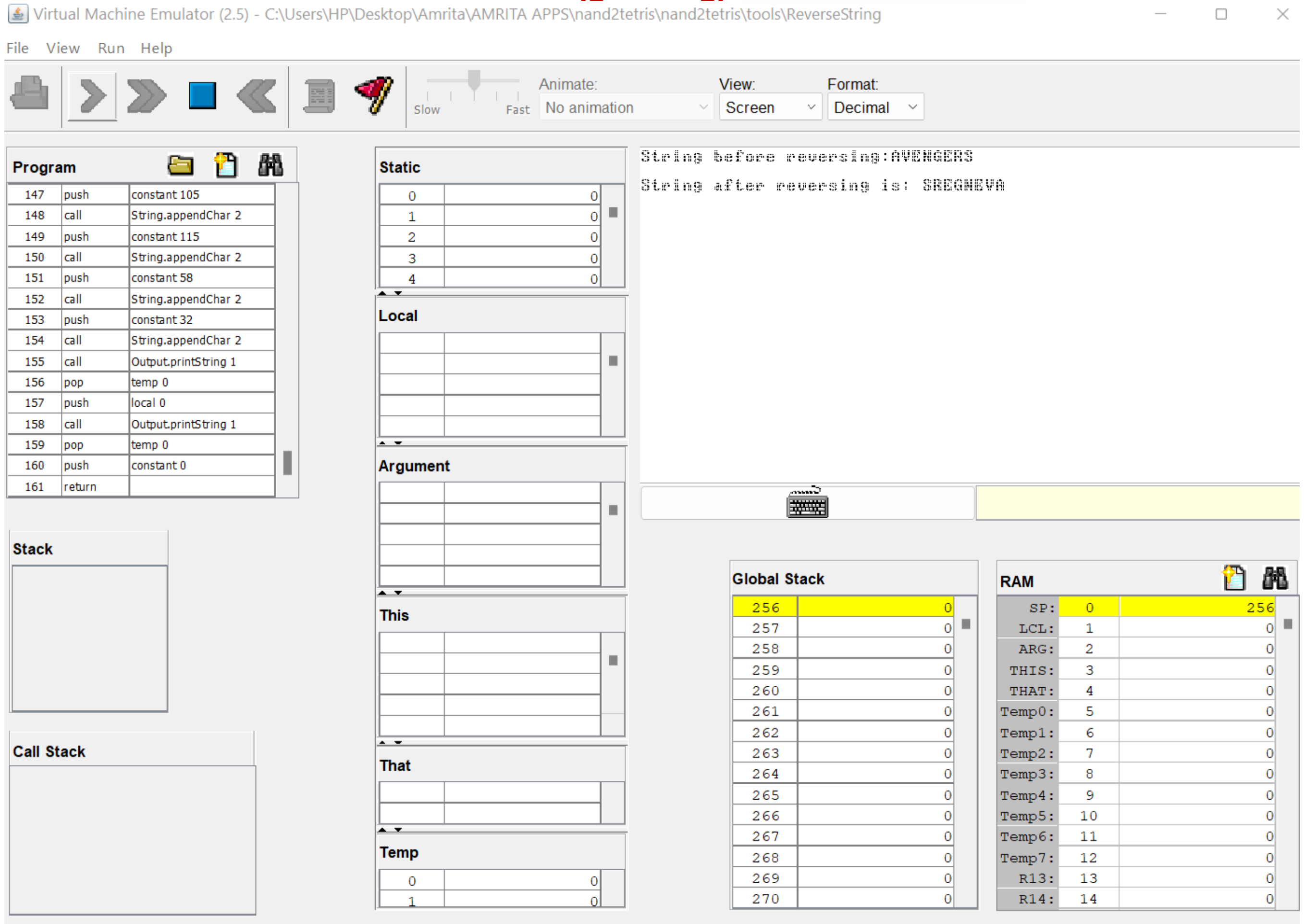
SP:	0	256
LCL:	1	0
ARG:	2	0
THIS:	3	0
THAT:	4	0
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

ALT-CODE



```
class Main{
    function void main(){
        var String inputString;
        var int i, j, length;
        var char tmpi, tmpj;
        let inputString = Keyboard.readLine("String before reversing:");
        do Output.println();

        let length = inputString.length() - 1;
        let j = length;
        let i = 0;
        while(~(j = (length/2))){
          let tmpi = inputString.charAt(i);
          let tmpj = inputString.charAt(j);
          do inputString.setCharAt(i, tmpj);
          do inputString.setCharAt(j, tmpi);
          let j = j - 1;
          let i = i + 1;
        }
        do Output.printString("String after reversing is: ");
        do Output.printString(inputString);
        return;
    }
}
```

Compiler



Virtual Machine Emulator (2.5) - C:\Users\HP\Desktop\Amrita\AMRITA APPS\nand2tetris\nand2tetris\tools\ReverseString

Animate: No animation View: Screen Format: Decimal

Program

147	push	constant 105
148	call	String.appendChar 2
149	push	constant 115
150	call	String.appendChar 2
151	push	constant 58
152	call	String.appendChar 2
153	push	constant 32
154	call	String.appendChar 2
155	call	Output.printString 1
156	pop	temp 0
157	push	local 0
158	call	Output.printString 1
159	pop	temp 0
160	push	constant 0
161	return	

Static

0	0
1	0
2	0
3	0
4	0

Temp

0	0
1	0

```
String before reversing:AVENGERS
String after reversing is: SREGNEVA
```

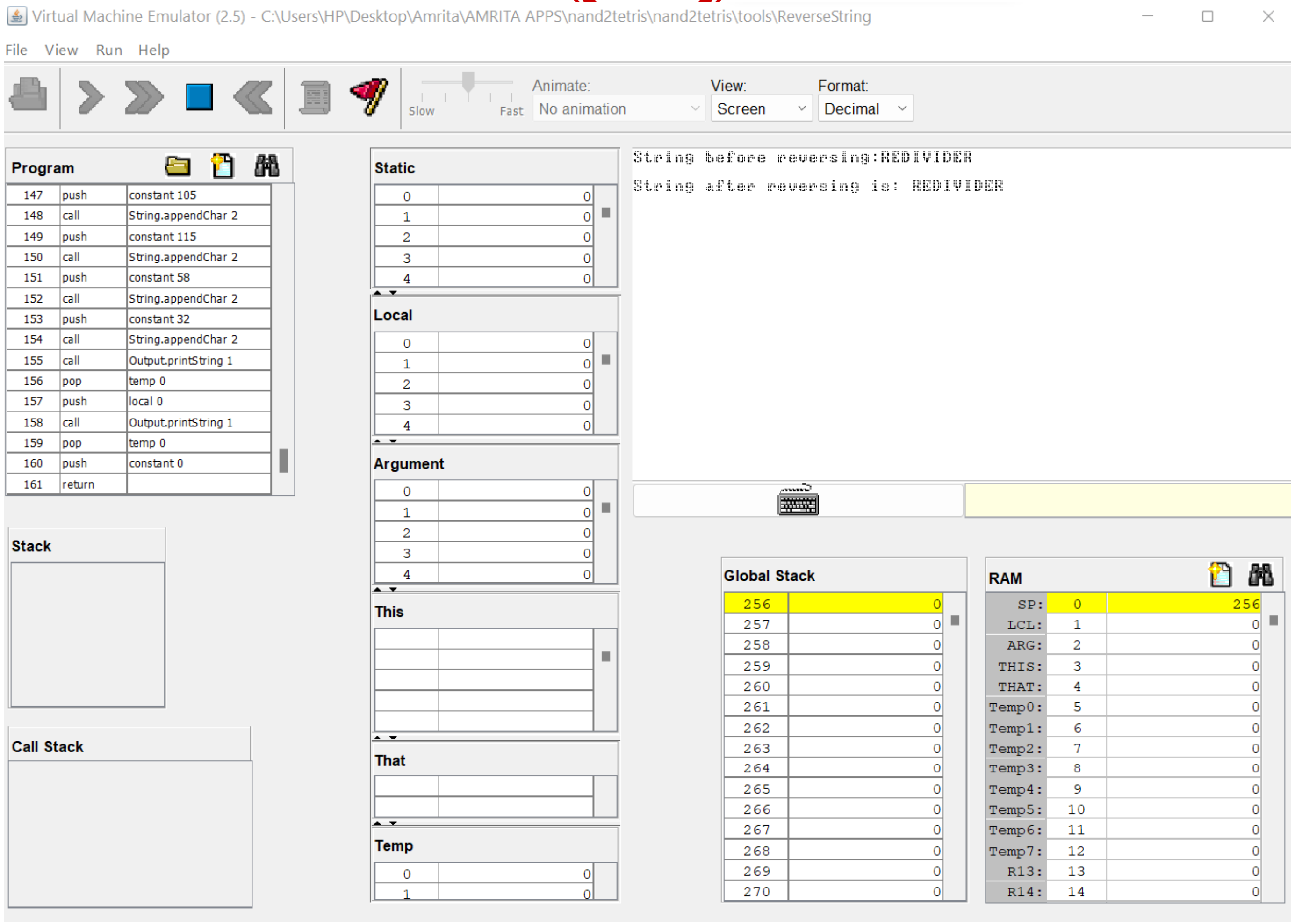
Global Stack

256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

RAM

SP:	0	256
LCL:	1	0
ARG:	2	0
THIS:	3	0
THAT:	4	0
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

OUTPUT



Virtual Machine Emulator (2.5) - C:\Users\HP\Desktop\Amrita\AMRITA APPS\nand2tetris\nand2tetris\tools\ReverseString

Animate: No animation
View: Screen
Format: Decimal

```
String before reversing:REDIVIDER
String after reversing is: REDIVIDER
```

Program

147	push	constant 105
148	call	String.appendChar 2
149	push	constant 115
150	call	String.appendChar 2
151	push	constant 58
152	call	String.appendChar 2
153	push	constant 32
154	call	String.appendChar 2
155	call	Output.printString 1
156	pop	temp 0
157	push	local 0
158	call	Output.printString 1
159	pop	temp 0
160	push	constant 0
161	return	

Static

0	0
1	0
2	0
3	0
4	0

Local

0	0
1	0
2	0
3	0
4	0

Argument

0	0
1	0
2	0
3	0
4	0

This

That

Temp

0	0
1	0

Stack

Call Stack

Global Stack

256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

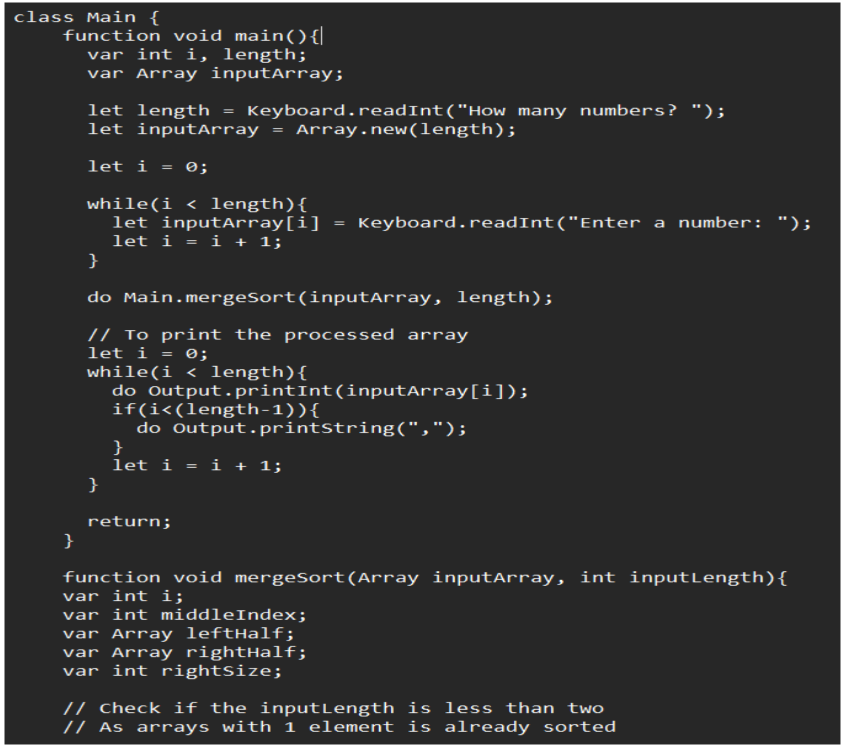
RAM

SP:	0	256
LCL:	1	0
ARG:	2	0
THIS:	3	0
THAT:	4	0
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

2

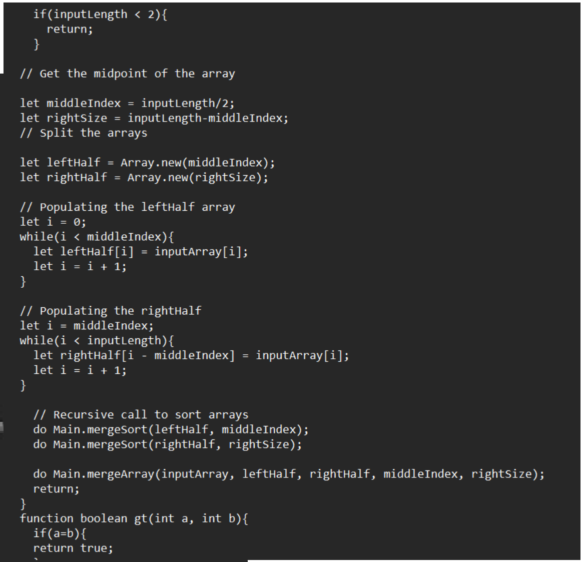
Write a JACK program to perform MERGE sort using recursion

CODE

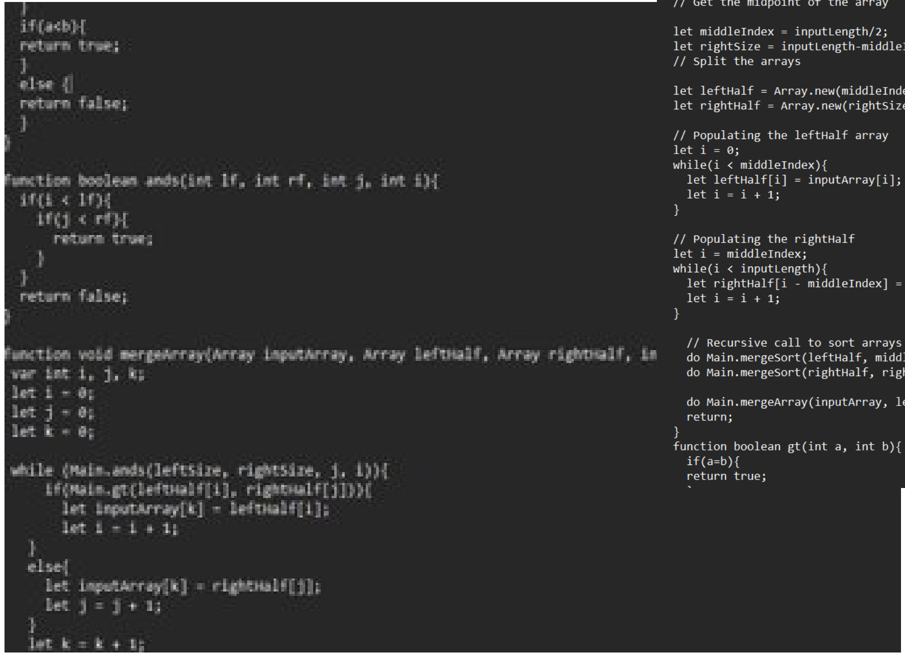


```
class Main {
    function void main(){
      var int i, length;
      var Array inputArray;

      let length = Keyboard.readInt("How many numbers? ");
      let inputArray = Array.new(length);

      let i = 0;

      while(i < length){
        let inputArray[i] = Keyboard.readInt("Enter a number: ");
        let i = i + 1;
      }

      do Main.mergeSort(inputArray, length);

      // To print the processed array
      let i = 0;
      while(i < length){
        do Output.printInt(inputArray[i]);
        if(i<(length-1)){
          do Output.printString(",");
        }
        let i = i + 1;
      }

      return;
    }

    function void mergeSort(Array inputArray, int inputLength){
    var int i;
    var int middleIndex;
    var Array leftHalf;
    var Array rightHalf;
    var int rightSize;

    // Check if the inputLength is less than two
    // As arrays with 1 element is already sorted
```

OUTPUT

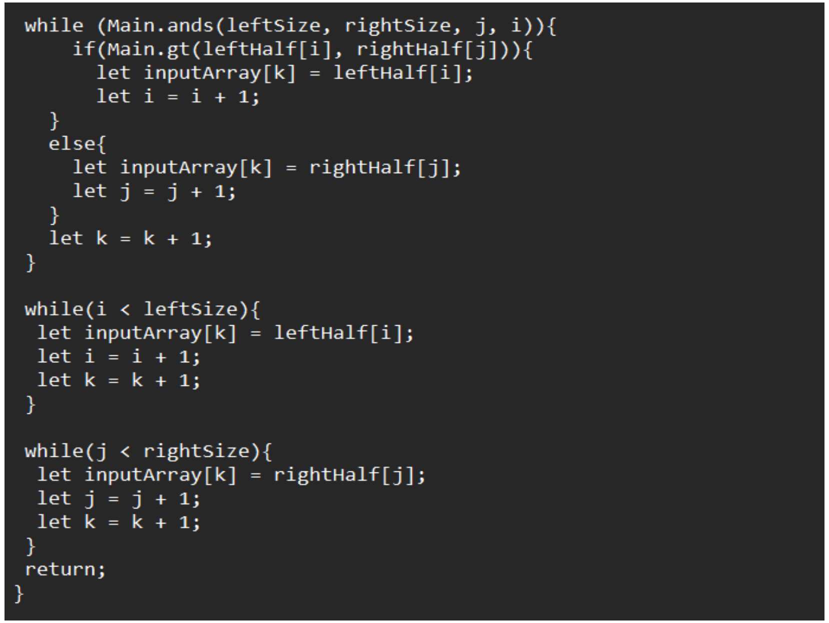


```
  if(inputLength < 2){
    return;
  }

// Get the midpoint of the array

let middleIndex = inputLength/2;
let rightSize = inputLength-middleIndex;
// Split the arrays

let leftHalf = Array.new(middleIndex);
let rightHalf = Array.new(rightSize);

// Populating the leftHalf array
let i = 0;
while(i < middleIndex){
  let leftHalf[i] = inputArray[i];
  let i = i + 1;
}

// Populating the rightHalf
let i = middleIndex;
while(i < inputLength){
  let rightHalf[i - middleIndex] = inputArray[i];
  let i = i + 1;
}

  // Recursive call to sort arrays
  do Main.mergeSort(leftHalf, middleIndex);
  do Main.mergeSort(rightHalf, rightSize);

  do Main.mergeArray(inputArray, leftHalf, rightHalf, middleIndex, rightSize);
  return;
}
function boolean gt(int a, int b){
  if(a=b){
  return true;
```



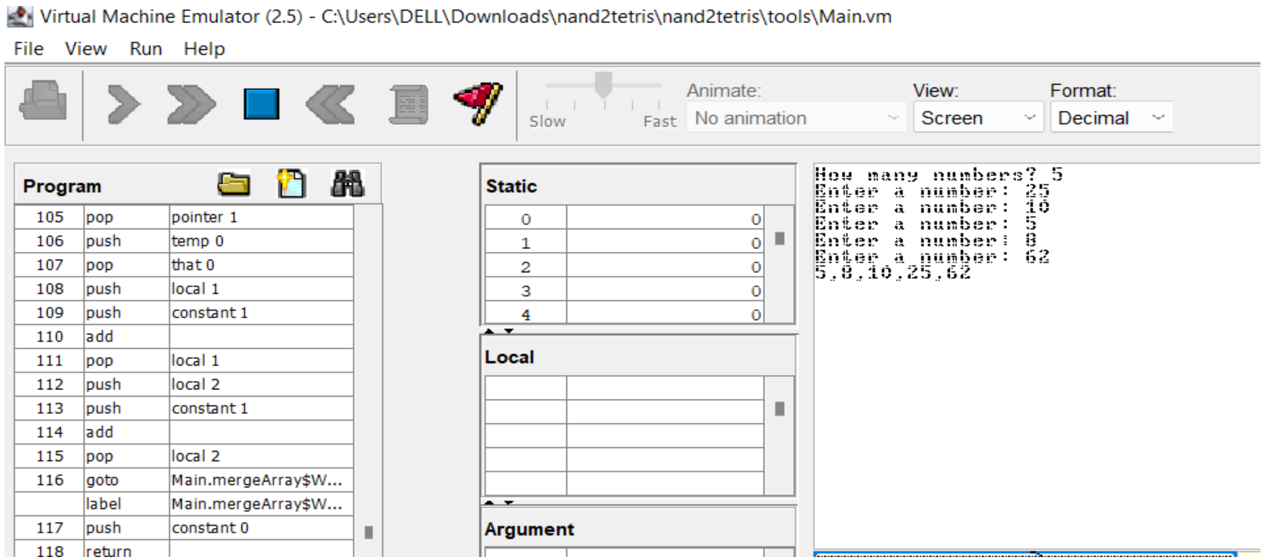
```
  }
  if(a<b){
  return true;
  }
  else {
  return false;
  }
}

function boolean ands(int lf, int rf, int j, int i){
  if(i < lf){
    if(j < rf){
      return true;
    }
  }
  return false;
}

function void mergeArray(Array inputArray, Array leftHalf, Array rightHalf, in
 var int i, j, k;
 let i = 0;
 let j = 0;
 let k = 0;

 while (Main.ands(leftSize, rightSize, j, i)){
     if(Main.gt(leftHalf[i], rightHalf[j])){
       let inputArray[k] = leftHalf[i];
       let i = i + 1;
   }
   else{
     let inputArray[k] = rightHalf[j];
     let j = j + 1;
   }
   let k = k + 1;
```

OUTPUT



```
 while (Main.ands(leftSize, rightSize, j, i)){
     if(Main.gt(leftHalf[i], rightHalf[j])){
       let inputArray[k] = leftHalf[i];
       let i = i + 1;
   }
   else{
     let inputArray[k] = rightHalf[j];
     let j = j + 1;
   }
   let k = k + 1;
 }

 while(i < leftSize){
  let inputArray[k] = leftHalf[i];
  let i = i + 1;
  let k = k + 1;
 }

 while(j < rightSize){
  let inputArray[k] = rightHalf[j];
  let j = j + 1;
  let k = k + 1;
 }
 return;
}
```

output



Virtual Machine Emulator (2.5) - C:\Users\DELL\Downloads\nand2tetris\nand2tetris\tools\Main.vm

Animate: No animation
View: Screen
Format: Decimal

Program

105	pop	pointer 1
106	push	temp 0
107	pop	that 0
108	push	local 1
109	push	constant 1
110	add	
111	pop	local 1
112	push	local 2
113	push	constant 1
114	add	
115	pop	local 2
116	goto	Main.mergeArray$W...
	label	Main.mergeArray$W...
117	push	constant 0
118	return	

Static

0	0
1	0
2	0
3	0
4	0

Local

Argument

```
How many numbers? 5
Enter a number: 25
Enter a number: 10
Enter a number: 5
Enter a number: 8
Enter a number: 62
5,8,10,25,62
```

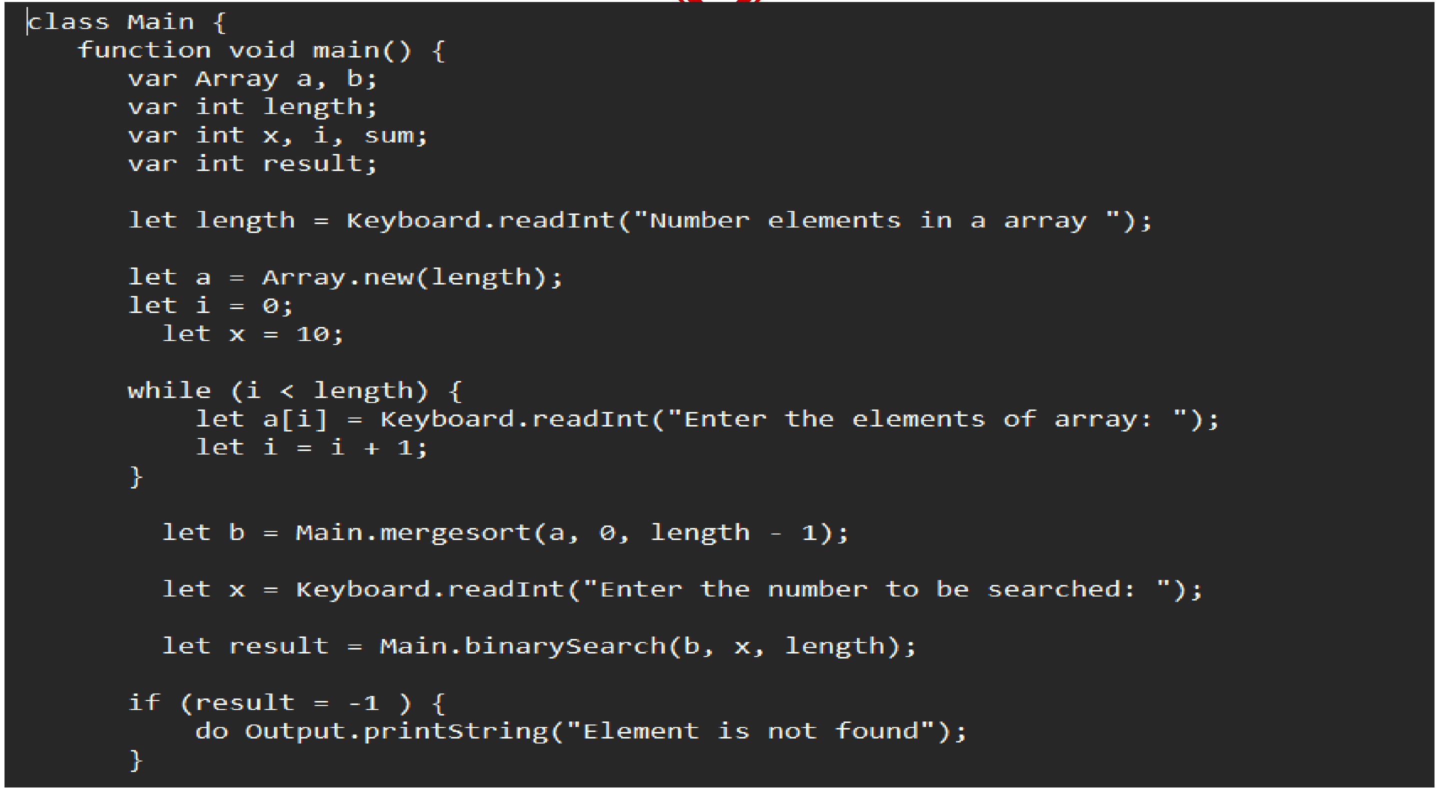
3

Write a JACK program to perform BINARY Search

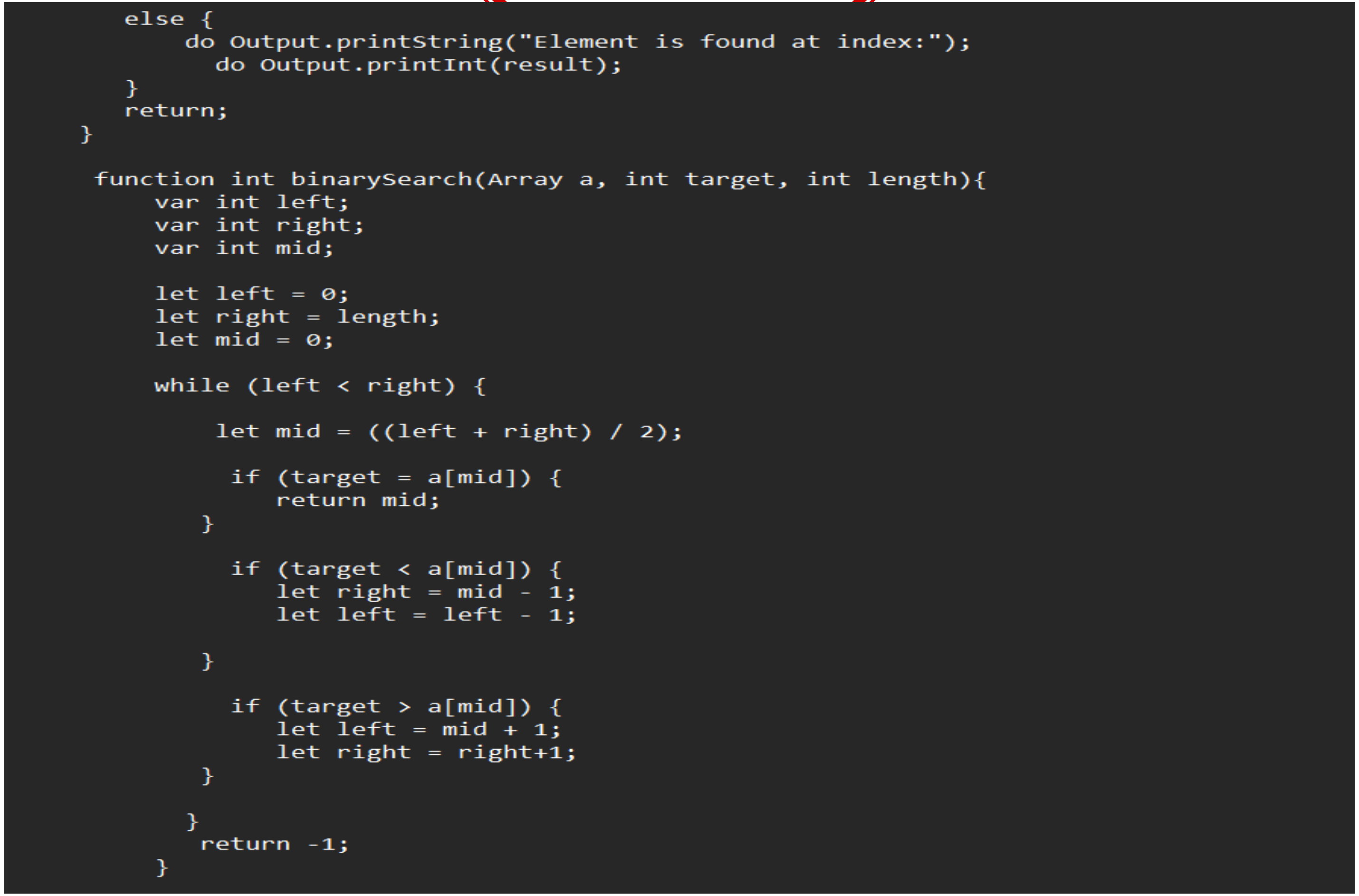
Binary Search

1)Another divide-and-conquer-based technique is binary search, which helps locate the index of an element in a sorted array.
2)Using the merge sort method we established, we can sort the array.
3)In a nutshell, binary search is similar to looking up a word in a dictionary. Because the dictionary is arranged alphabetically.
4)We can look for the letter by taking half of it and then checking if the letter matches if it is discovered.

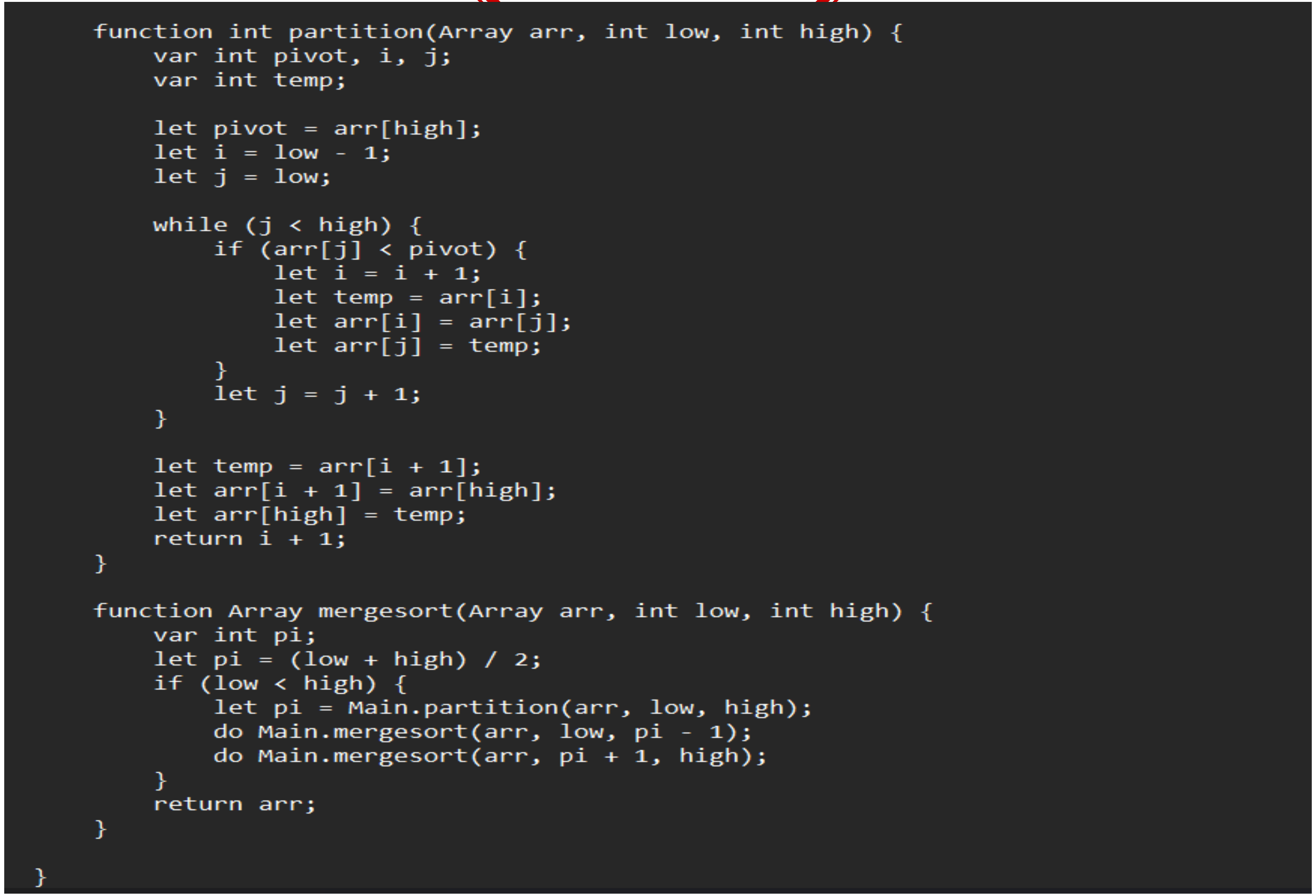
Jack code



```
class Main {
   function void main() {
      var Array a, b;
      var int length;
      var int x, i, sum;
      var int result;

      let length = Keyboard.readInt("Number elements in a array ");

      let a = Array.new(length);
      let i = 0;
        let x = 10;

      while (i < length) {
          let a[i] = Keyboard.readInt("Enter the elements of array: ");
          let i = i + 1;
      }

        let b = Main.mergesort(a, 0, length - 1);

        let x = Keyboard.readInt("Enter the number to be searched: ");

        let result = Main.binarySearch(b, x, length);

      if (result = -1 ) {
          do Output.printString("Element is not found");
      }
```

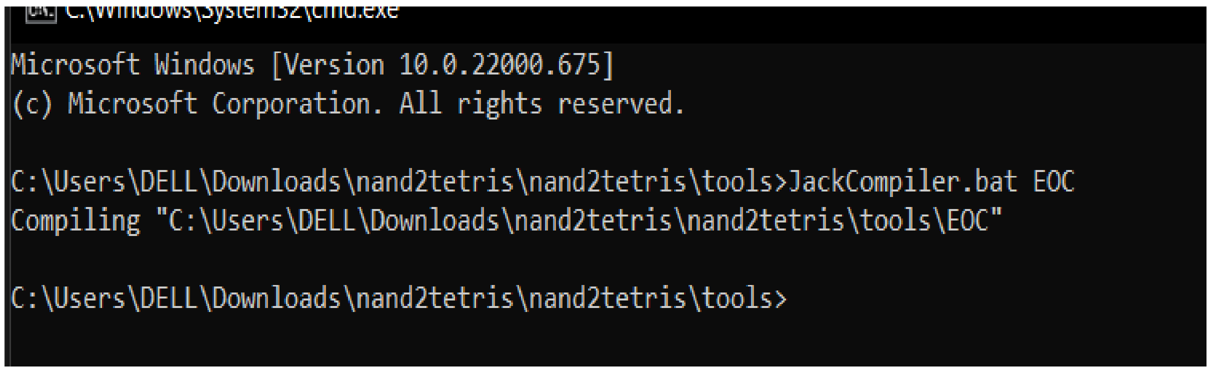


```
   else {
       do Output.printString("Element is found at index:");
         do Output.printInt(result);
   }
   return;
}

 function int binarySearch(Array a, int target, int length){
     var int left;
     var int right;
     var int mid;

     let left = 0;
     let right = length;
     let mid = 0;

     while (left < right) {

         let mid = ((left + right) / 2);

          if (target = a[mid]) {
             return mid;
        }

          if (target < a[mid]) {
             let right = mid - 1;
             let left = left - 1;

        }

          if (target > a[mid]) {
             let left = mid + 1;
             let right = right+1;
        }

       }
        return -1;
     }
```



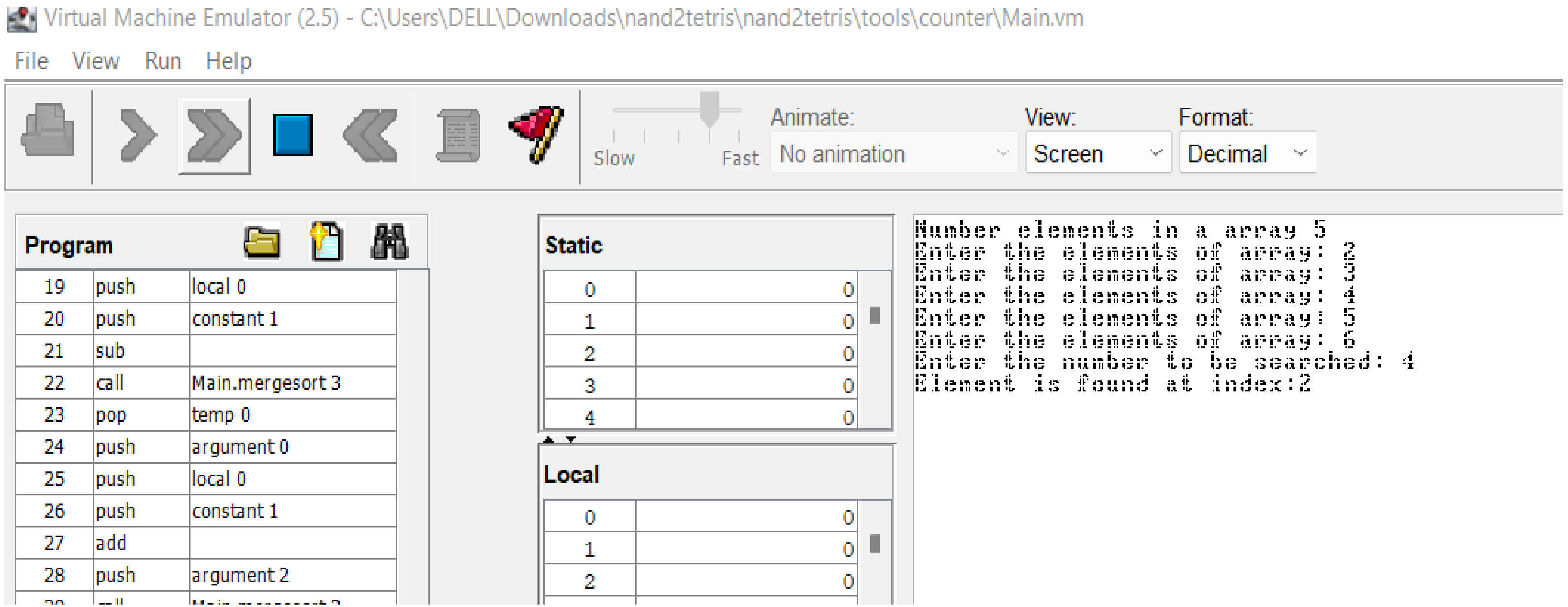
```
    function int partition(Array arr, int low, int high) {
        var int pivot, i, j;
        var int temp;

        let pivot = arr[high];
        let i = low - 1;
        let j = low;

        while (j < high) {
            if (arr[j] < pivot) {
                let i = i + 1;
                let temp = arr[i];
                let arr[i] = arr[j];
                let arr[j] = temp;
            }
            let j = j + 1;
        }

        let temp = arr[i + 1];
        let arr[i + 1] = arr[high];
        let arr[high] = temp;
        return i + 1;
    }

    function Array mergesort(Array arr, int low, int high) {
        var int pi;
        let pi = (low + high) / 2;
        if (low < high) {
            let pi = Main.partition(arr, low, high);
            do Main.mergesort(arr, low, pi - 1);
            do Main.mergesort(arr, pi + 1, high);
        }
        return arr;
    }

}
```

Next we have to compile that in command prompt



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.675]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL\Downloads\nand2tetris\nand2tetris\tools>JackCompiler.bat EOC
Compiling "C:\Users\DELL\Downloads\nand2tetris\nand2tetris\tools\EOC"

C:\Users\DELL\Downloads\nand2tetris\nand2tetris\tools>
```

OUTPUT


Virtual Machine Emulator (2.5) - C:\Users\DELL\Downloads\nand2tetris\nand2tetris\tools\counter\Main.vm
File View Run Help
Slow Fast
Animate: No animation
View: Screen
Format: Decimal
Program
Static
Local

19	push	local 0
20	push	constant 1
21	sub	
22	call	Main.mergesort 3
23	pop	temp 0
24	push	argument 0
25	push	local 0
26	push	constant 1
27	add	
28	push	argument 2

Static

0	0
1	0
2	0
3	0
4	0

Local

0	0
1	0
2	0

```
Number elements in a array 5
Enter the elements of array: 2
Enter the elements of array: 3
Enter the elements of array: 4
Enter the elements of array: 5
Enter the elements of array: 6
Enter the number to be searched: 4
Element is found at index:2
```

4

Write a JACK program to find the smallest and largest number in an array.

Logic

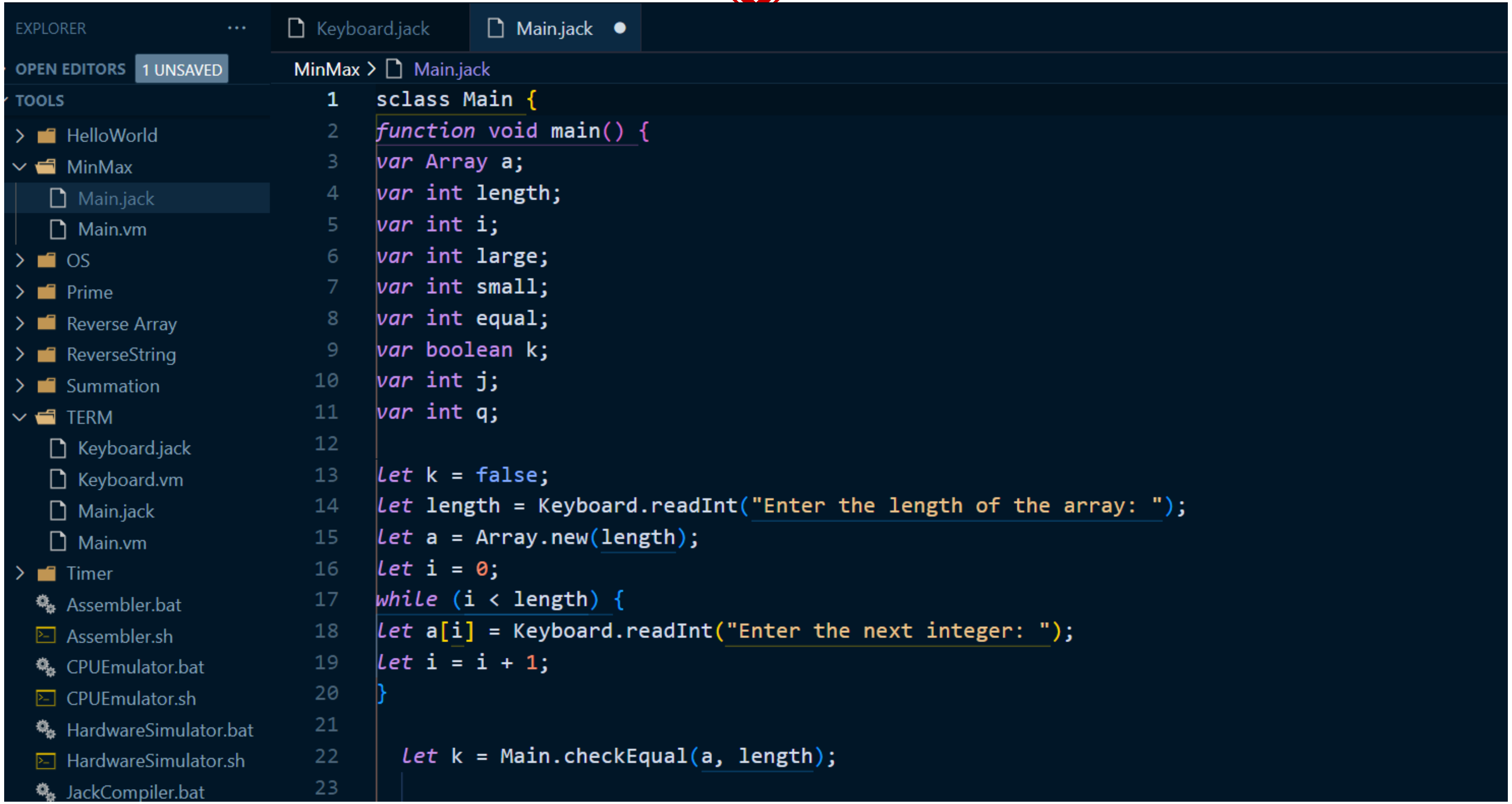
We will take the length of an array as an input from user.

We will take the elements of the array as an input from the user.

We will create a separate function which will check whether our array contains equal elements or not.

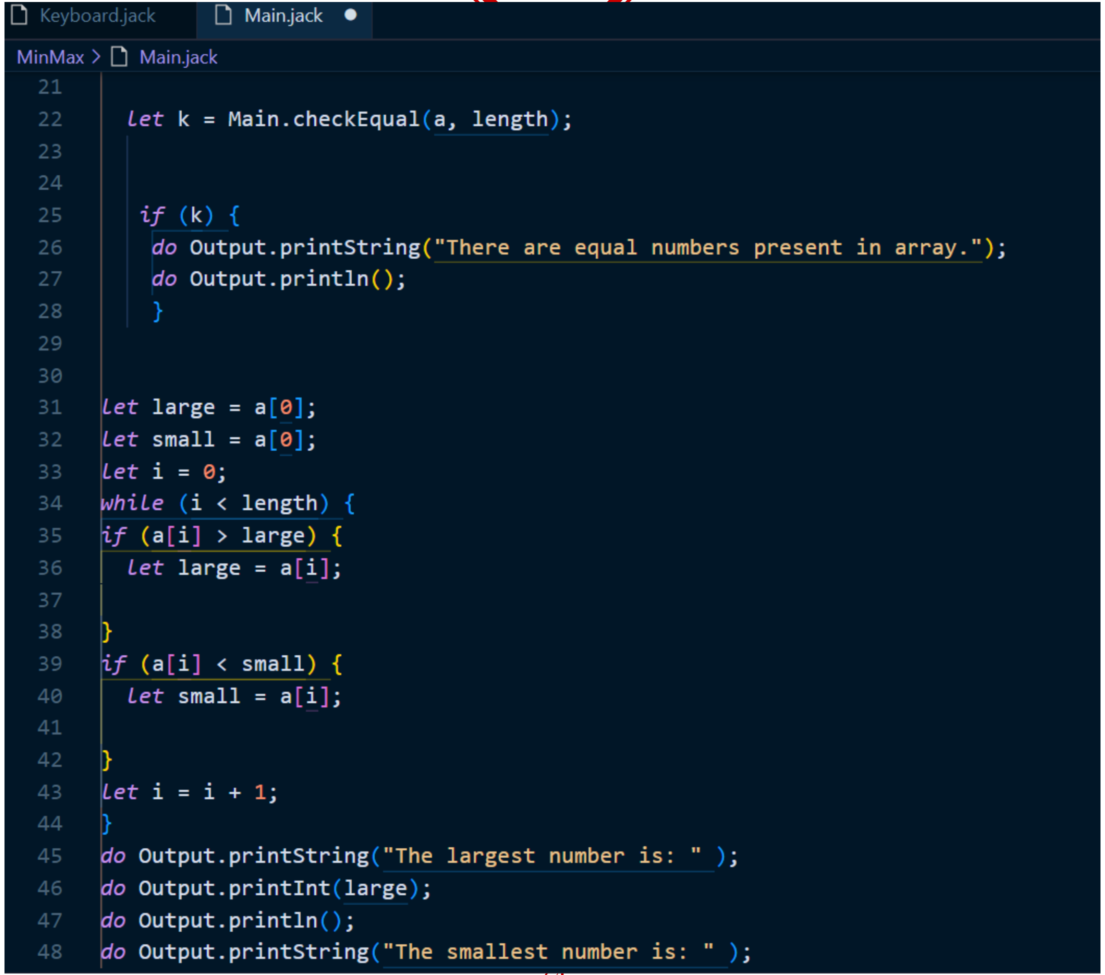
Finally, we will create a main function which will contain the while loop and if-condition which will check the smallest and largest array.

CODE



```
sclass Main {
function void main() {
var Array a;
var int length;
var int i;
var int large;
var int small;
var int equal;
var boolean k;
var int j;
var int q;

let k = false;
let length = Keyboard.readInt("Enter the length of the array: ");
let a = Array.new(length);
let i = 0;
while (i < length) {
let a[i] = Keyboard.readInt("Enter the next integer: ");
let i = i + 1;
}

  let k = Main.checkEqual(a, length);

```

CODE

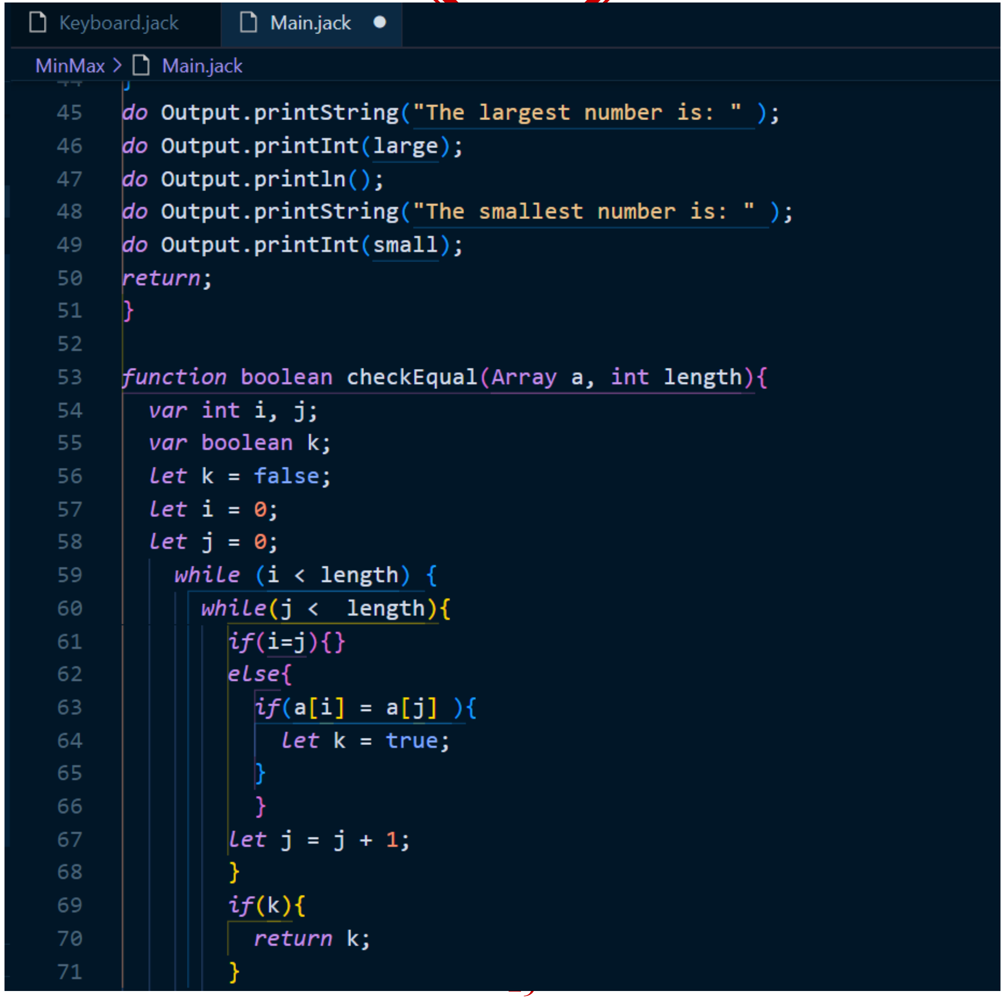


Keyboard.jack   Main.jack

MinMax > Main.jack

```

  let k = Main.checkEqual(a, length);


   if (k) {
    do Output.printString("There are equal numbers present in array.");
    do Output.println();
    }


let large = a[0];
let small = a[0];
let i = 0;
while (i < length) {
if (a[i] > large) {
  let large = a[i];

}
if (a[i] < small) {
  let small = a[i];

}
let i = i + 1;
}
do Output.printString("The largest number is: " );
do Output.printInt(large);
do Output.println();
do Output.printString("The smallest number is: " );
```

CODE

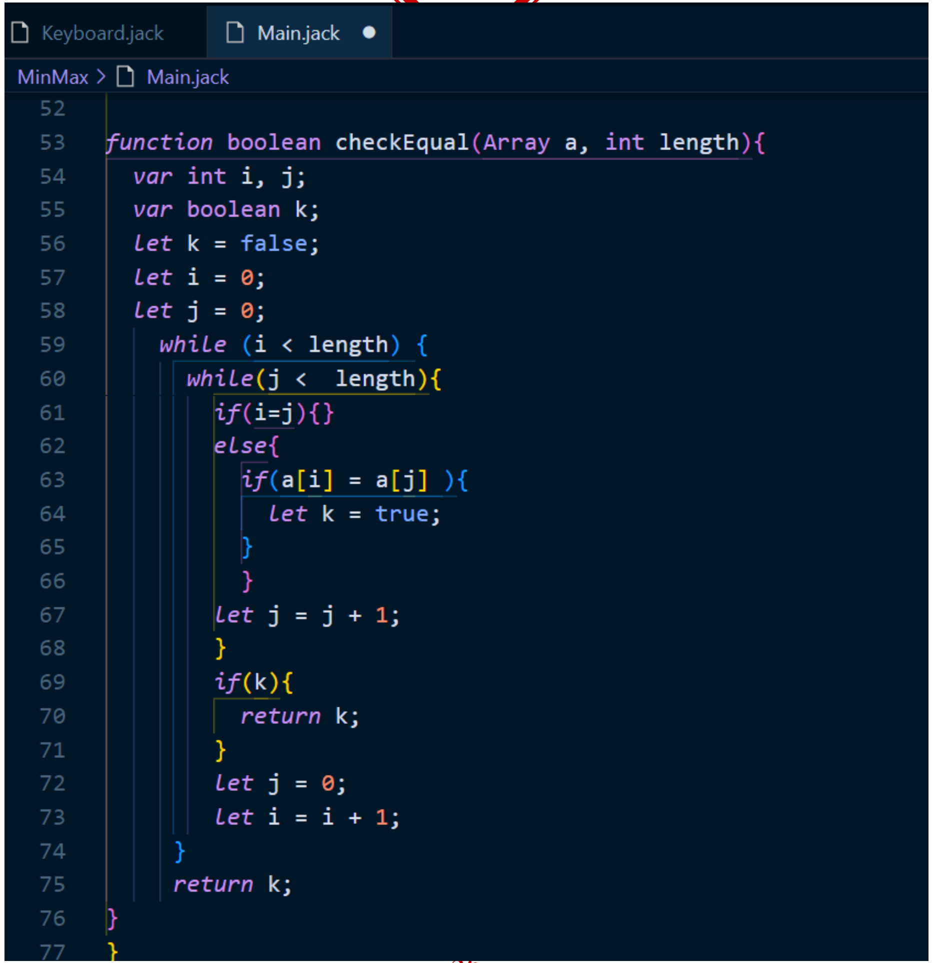


Keyboard.jack   Main.jack

MinMax > Main.jack

```
do Output.printString("The largest number is: " );
do Output.printInt(large);
do Output.println();
do Output.printString("The smallest number is: " );
do Output.printInt(small);
return;
}

function boolean checkEqual(Array a, int length){
  var int i, j;
  var boolean k;
  let k = false;
  let i = 0;
  let j = 0;
    while (i < length) {
      while(j <  length){
        if(i=j){}
        else{
          if(a[i] = a[j] ){
            let k = true;
          }
          }
        let j = j + 1;
        }
        if(k){
          return k;
        }
```

CODE

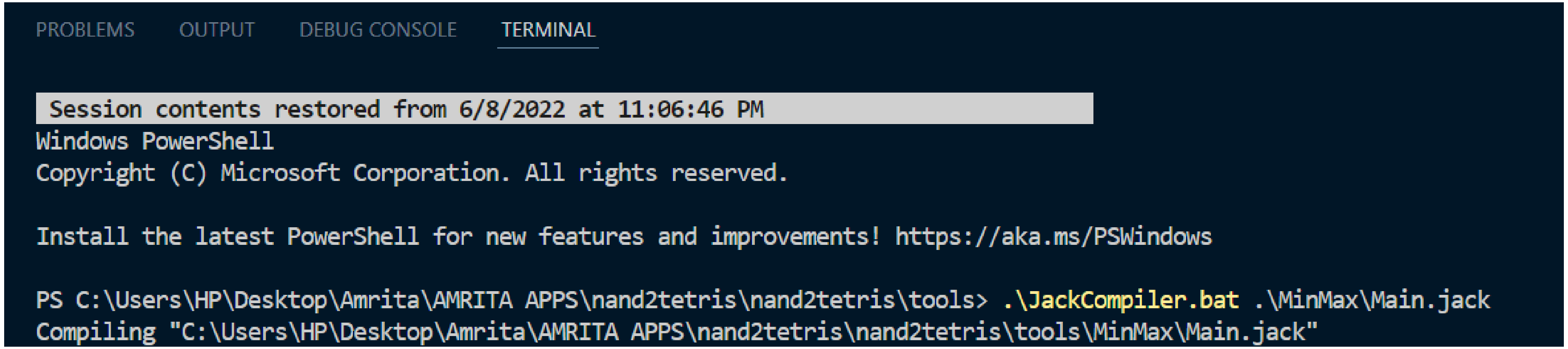


Keyboard.jack   Main.jack

MinMax > Main.jack

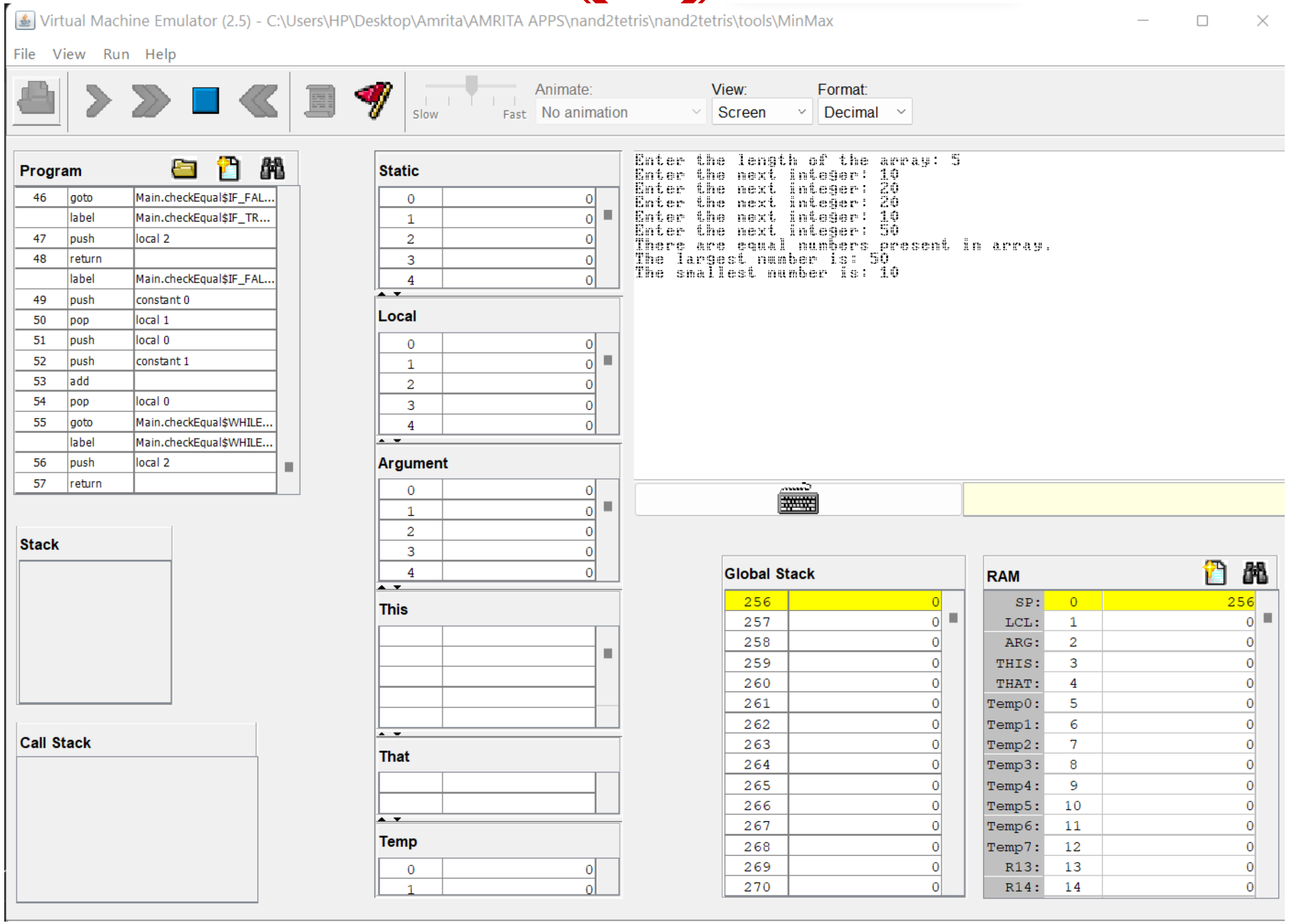
```

function boolean checkEqual(Array a, int length){
  var int i, j;
  var boolean k;
  let k = false;
  let i = 0;
  let j = 0;
    while (i < length) {
      while(j <  length){
        if(i=j){}
        else{
          if(a[i] = a[j] ){
            let k = true;
          }
          }
        let j = j + 1;
        }
        if(k){
          return k;
        }
        let j = 0;
        let i = i + 1;
    }
    return k;
}
}
```

Compile



```
 Session contents restored from 6/8/2022 at 11:06:46 PM
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\HP\Desktop\Amrita\AMRITA APPS\nand2tetris\nand2tetris\tools> .\JackCompiler.bat .\MinMax\Main.jack
Compiling "C:\Users\HP\Desktop\Amrita\AMRITA APPS\nand2tetris\nand2tetris\tools\MinMax\Main.jack"
```

OUTPUT



Virtual Machine Emulator (2.5) - C:\Users\HP\Desktop\Amrita\AMRITA APPS\nand2tetris\nand2tetris\tools\MinMax

Animate: No animation View: Screen Format: Decimal

Program

46	goto	Main.checkEqual$IF_FAL...
	label	Main.checkEqual$IF_TR...
47	push	local 2
48	return	
	label	Main.checkEqual$IF_FAL...
49	push	constant 0
50	pop	local 1
51	push	local 0
52	push	constant 1
53	add	
54	pop	local 0
55	goto	Main.checkEqual$WHILE...
	label	Main.checkEqual$WHILE...
56	push	local 2
57	return	

```
Enter the length of the array: 5
Enter the next integer: 10
Enter the next integer: 20
Enter the next integer: 20
Enter the next integer: 10
Enter the next integer: 50
There are equal numbers present in array.
The largest number is: 50
The smallest number is: 10
```

Stack

Call Stack

Static

0	0
1	0
2	0
3	0
4	0

Local

0	0
1	0
2	0
3	0
4	0

Argument

0	0
1	0
2	0
3	0
4	0

This

That

Temp

0	0
1	0

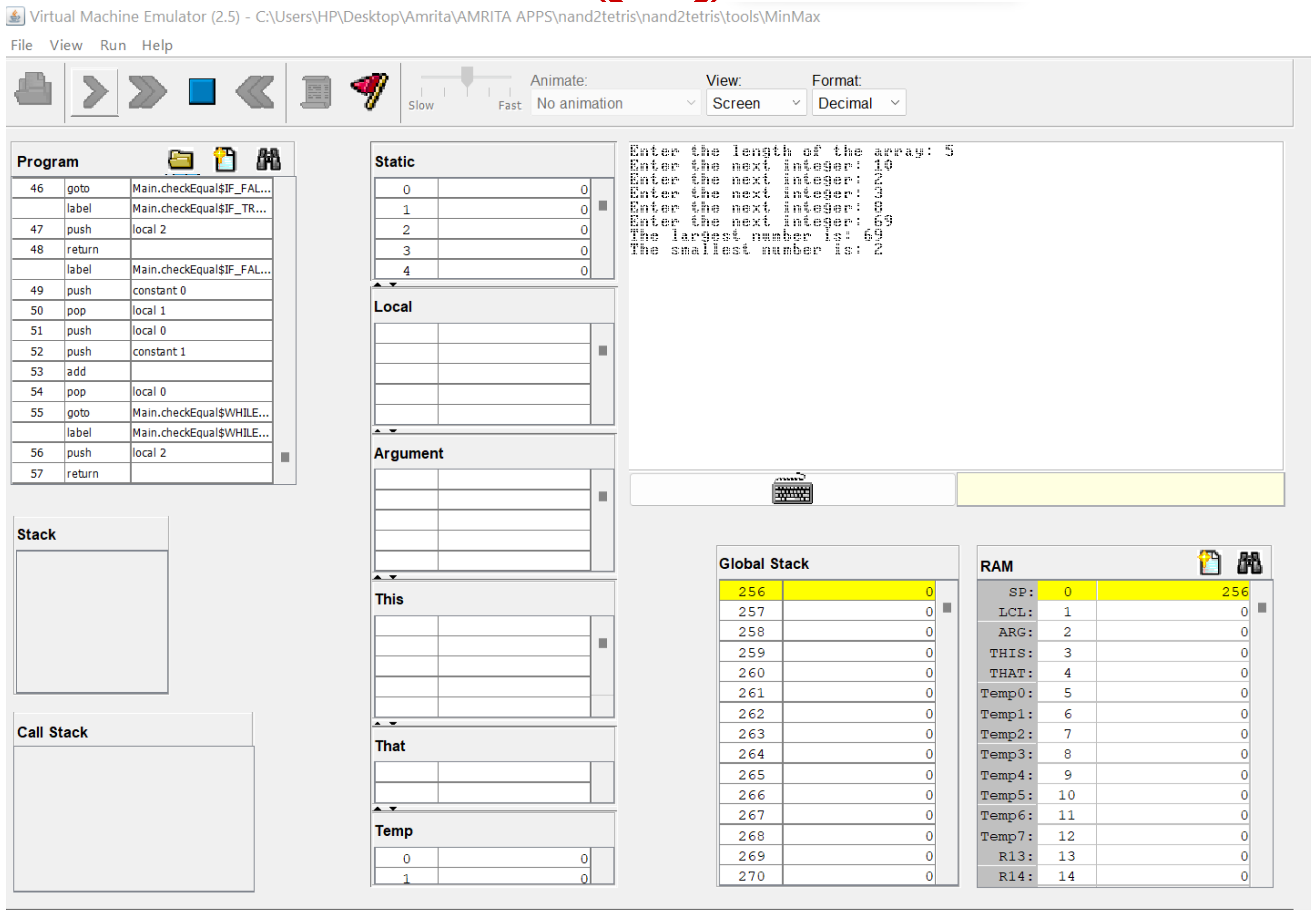
Global Stack

256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

RAM

SP:	0	256
LCL:	1	0
ARG:	2	0
THIS:	3	0
THAT:	4	0
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

OUTPUT



46	goto	Main.checkEqual$IF_FAL...
	label	Main.checkEqual$IF_TR...
47	push	local 2
48	return	
	label	Main.checkEqual$IF_FAL...
49	push	constant 0
50	pop	local 1
51	push	local 0
52	push	constant 1
53	add	
54	pop	local 0
55	goto	Main.checkEqual$WHILE...
	label	Main.checkEqual$WHILE...
56	push	local 2
57	return	

Static

0	0
1	0
2	0
3	0
4	0

Temp

0	0
1	0

```
Enter the length of the array: 5
Enter the next integer: 10
Enter the next integer: 2
Enter the next integer: 3
Enter the next integer: 8
Enter the next integer: 69
The largest number is: 69
The smallest number is: 2
```

Global Stack

256	0
257	0
258	0
259	0
260	0
261	0
262	0
263	0
264	0
265	0
266	0
267	0
268	0
269	0
270	0

RAM

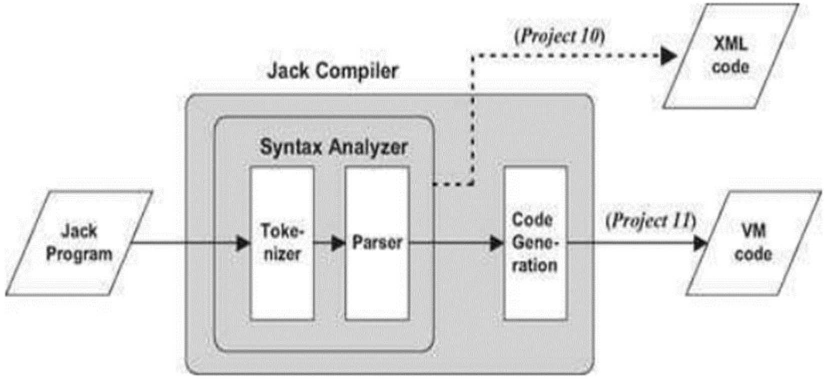
SP:	0	256
LCL:	1	0
ARG:	2	0
THIS:	3	0
THAT:	4	0
Temp0:	5	0
Temp1:	6	0
Temp2:	7	0
Temp3:	8	0
Temp4:	9	0
Temp5:	10	0
Temp6:	11	0
Temp7:	12	0
R13:	13	0
R14:	14	0

5

Give a detailed description of JACK Syntax Analyzer. Explain various steps involved in Tokenizer and Parser including the API

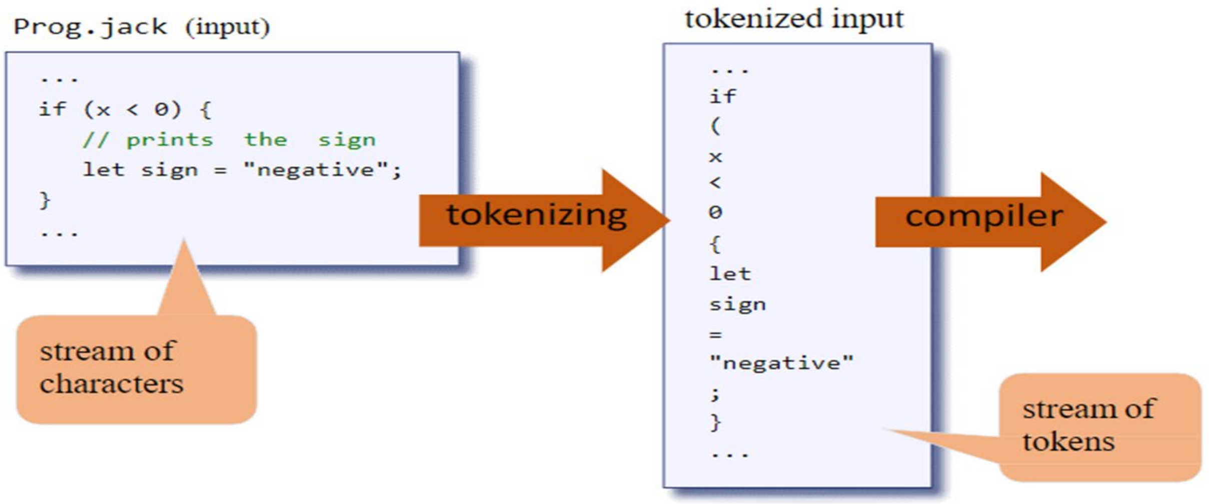
CODE

- Jack being a high-level language needs to be compiled into VMCode (then to assembly).
- Various steps are undergone till the compiler can output VMCode.
- Programming languages are usually described using a set of rules called context-free grammar.



- Syntax analyzer is one of the integral part in converting the high-level Jack code to VMCode.
- The syntax analyzer has mainly two part:
  - Tokenizer
  - Parser
- The Tokenizer groups characters into tokens.
- The parser has a set of rules which help it to parse the code.

Tokenizer:



```
...
if (x < 0) {
   // prints  the  sign
   let sign = "negative";
}
...
```

```
...
if
(
x
<
0
{
let
sign
=
"negative"
;
}
...
```

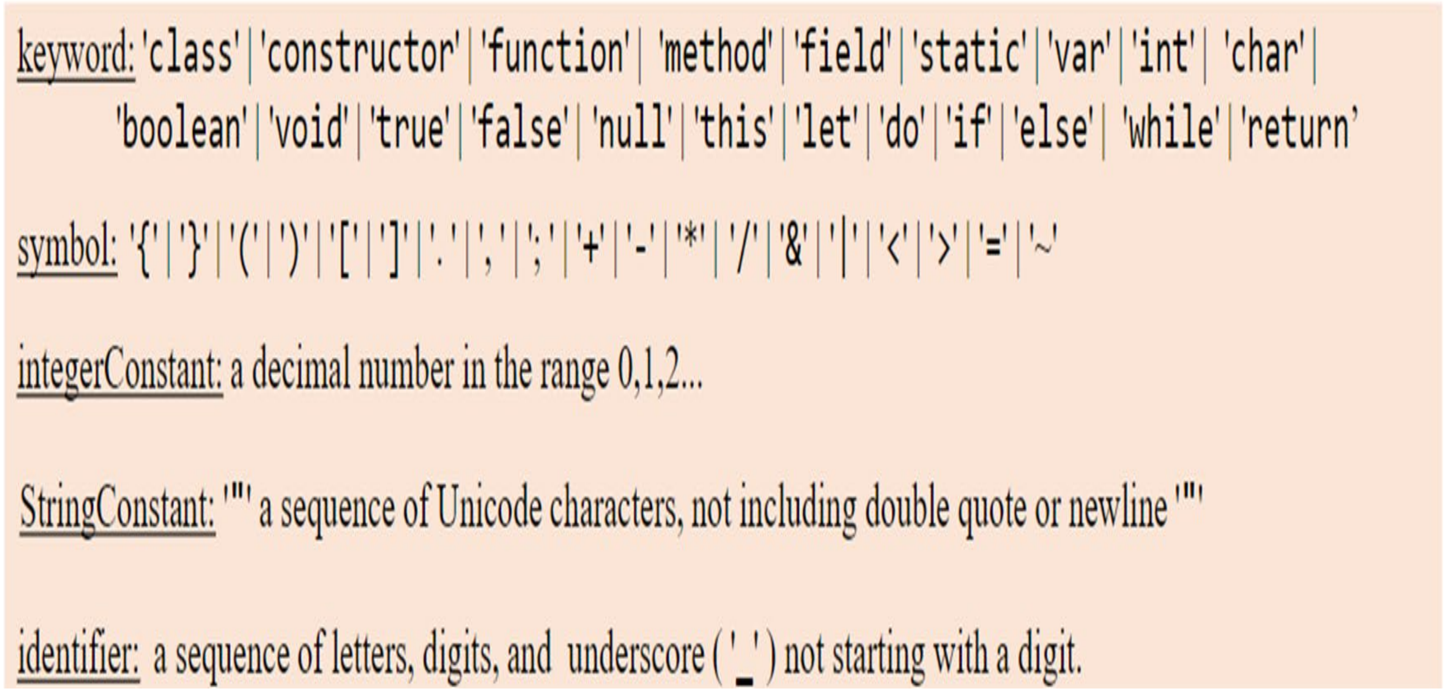
Tokenizing = grouping characters into tokens

A *token* is a string of characters that has a meaning

A programming language specification must document (among other things) its allowable tokens.

- The first step in the syntax analysis of a program is to group the characters into tokens (as defined by the language syntax), while ignoring white space and comments.
- This step is usually called lexical analysis, scanning, or tokenizing.
- Once a program has been tokenized, the tokens (rather than the characters) are viewed as its core component and the tokens stream becomes the main input of the compiler.

- Jack Tokens include:
  - Keywords
  - Symbols
  - Integers
  - Strings
  - Identifiers



keyword: 'class' | 'constructor' | 'function' | 'method' | 'field' | 'static' | 'var' | 'int' | 'char' |
'boolean' | 'void' | 'true' | 'false' | 'null' | 'this' | 'let' | 'do' | 'if' | 'else' | 'while' | 'return'

symbol: '{' | '}' | '(' | ')' | '[' | ']' | '.' | ',' | ';' | '+' | '-' | '*' | '/' | '&' | '|' | '<' | '>' | '=' | '~'

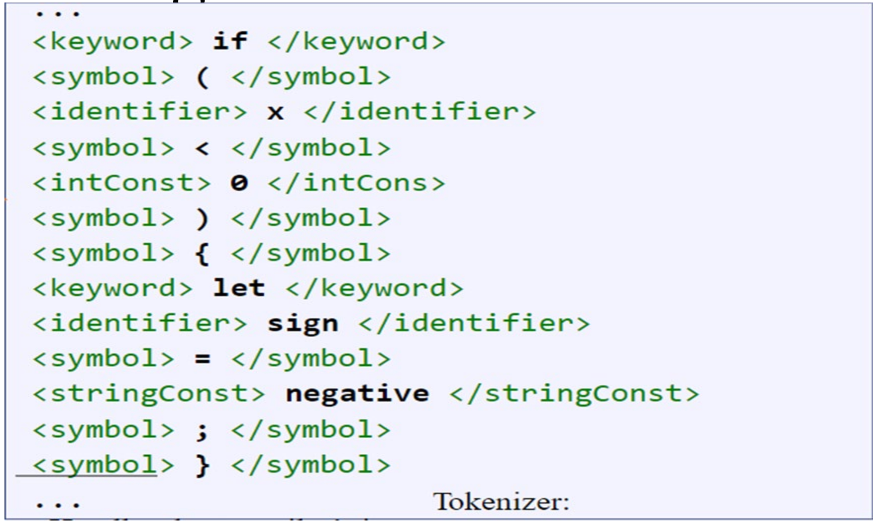
integerConstant: a decimal number in the range 0,1,2...

StringConstant: '"' a sequence of Unicode characters, not including double quote or newline '"'

identifier: a sequence of letters, digits, and underscore ('_') not starting with a digit.

After Tokenization

- As observable, each of the language 'items' has been classified, tokenized.
- This step also helps to verify the correctness of the compiled program by checking the XML file.



```
...
<keyword> if </keyword>
<symbol> ( </symbol>
<identifier> x </identifier>
<symbol> < </symbol>
<intConst> 0 </intCons>
<symbol> ) </symbol>
<symbol> { </symbol>
<keyword> let </keyword>
<identifier> sign </identifier>
<symbol> = </symbol>
<stringConst> negative </stringConst>
<symbol> ; </symbol>
<symbol> } </symbol>
...
```

Tokenizer:

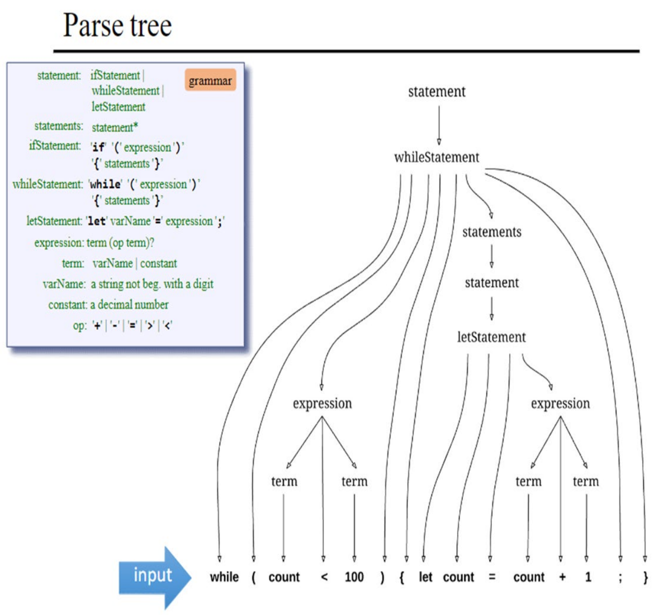
PRASER

- The act of checking whether a grammar "accepts" an input text as valid is called parsing.
- A grammar is a set of rules, describing how tokens can be combined to create valid language constructs.
- Since the grammar rules are hierarchical, the output generated by the parser can be described in a tree-oriented data structure called a parse tree or a derivation tree.

- The parser that Jack uses (included with Nand2tetris) represent the program’s structure implicitly, generate code and reporting errors on the fly.
- Such compilers don’t have to hold the entire program structure in memory, but only the subtree associated with the presently parsed element.

PARSE TREE

- The parse tree is the representation of the "Recursive Descent Parsing" method employed.
- The parser consists of a set of compile xxxmethods
- Each compile xxx method implements the right-hand side of the grammar rule describing xxx.



```
     statement:  ifStatement |
                 whileStatement |
                 letStatement
    statements:  statement*
   ifStatement:  'if' '(' expression ')'
                 '{' statements '}'
whileStatement:  'while' '(' expression ')'
                 '{' statements '}'
  letStatement:  'let' varName '=' expression ';'
    expression:  term (op term)?
          term:  varName | constant
       varName:  a string not beg. with a digit
      constant:  a decimal number
            op:  '+' | '-' | '=' | '>' | '<'
```

JACK SYNTAX ANALYZER

- The Syntax analyzer when run in a directory with .jack files will translate it to readable .XML files and save it to the same directory and also maintains the file name.
- Each .jack file is a stream of character, and these characters are tokenized by the Syntax Analyzer.
- The tokens may be separated by an arbitrary number of space characters, newline characters, and comments, which are ignored.

- Comments are of the standard formats /* comment until closing /, /* API comment */, and // comment to end of line.
- The Syntax Analyzer has three modules:

Jack Analyzer: top-level driver that sets up and invokes the other modules.

Jack Tokenizer: tokenizer.

Compilation Engine: recursive top-down parser.

The Jack Tokenizer ignores all comments and white spaces in the input stream and serializes it into Jack-Language tokens.

CONSTRUCTOR

- **Arguments:** Input File/Stream.
- **Function:** Opens the input file/stream and gets ready to tokenize it.

hasMoreTokens

- Checks if there is more tokens in the input and returns a Boolean.
- Returns true if there is more tokens else returns false.

advance

- Gets the next token from the input and makes it the current token.
- This method should only be called if hasMoreTokens() is true.
- Initially there is no current token.
- **Token Type**
- Returns the type of current token.

- **Return types:**
- Keyword
- Symbol
- Identifier
- INT_CONST
- STRING_CONST

SYMBOL

- Returns the character which is the current token. Should be called only when tokentype() is SYMBOL.
- **Return Type:**

Char

- **Identifier**

Returns the identifier which is the current token. Should be called only when tokenType() is IDENTIFIER

Return Type:

- String

intVal

- Returns the integer value of the current token should be called only when tokenType() is INT_CONST.

Return Type:

- Int

stringVal

- Returns the string value of the current token, without the double quotes Should be called only when tokenType() is STRING_CONST.

Return Type:

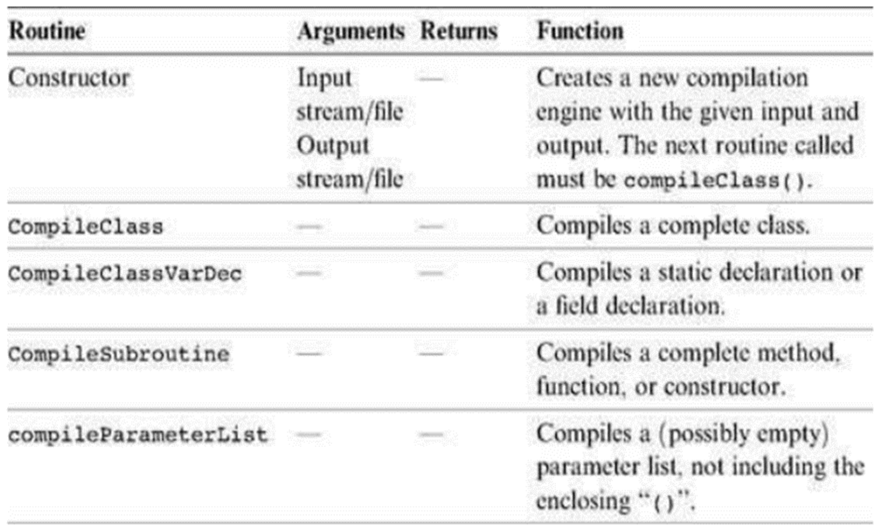
- String

COMPILATION ENGINE

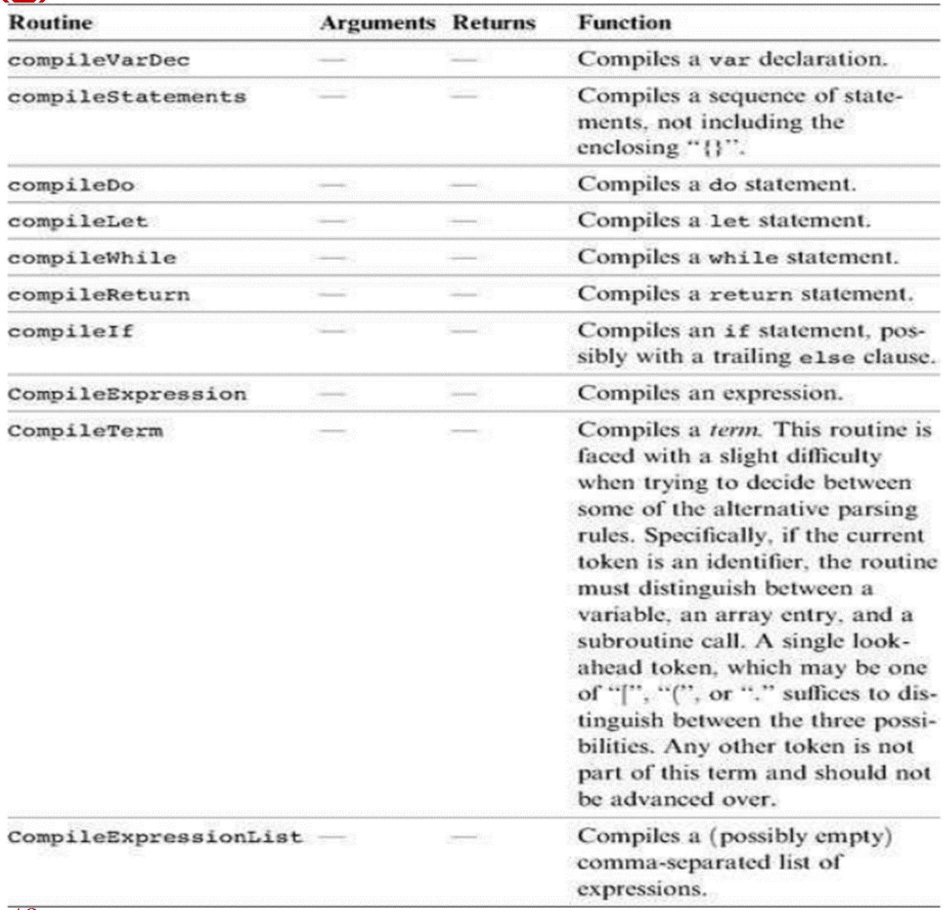
- Gets its input from a Jack Tokenizer and emits its parsed structure into an output file/stream.
- The output is generated by a series of compilexxx ( ) routines, one for every syntactic element xxx of the Jack grammar.
- The contract between these routines is that each compilexxx ( ) routine should read the syntactic construct xxx from the input, advance ( ) the tokenizer exactly beyond xxx, and output the parsing of xxx.

- This module emits a structured printout of the code, wrapped in XML tags. In the final version of the compiler, this module generates executable VM code.

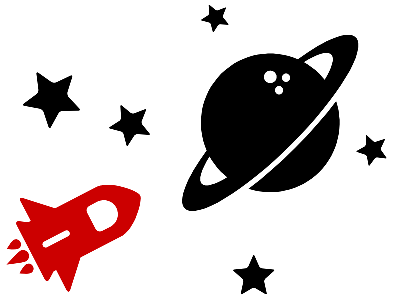
- Thus, compilexxx ( ) may only be called if indeed xxx is the next syntactic element of the input.



Routine	Arguments	Returns	Function
Constructor	Input stream/file Output stream/file	—	Creates a new compilation engine with the given input and output. The next routine called must be `compileClass()`.
`CompileClass`	—	—	Compiles a complete class.
`CompileClassVarDec`	—	—	Compiles a static declaration or a field declaration.
`CompileSubroutine`	—	—	Compiles a complete method, function, or constructor.
`compileParameterList`	—	—	Compiles a (possibly empty) parameter list, not including the enclosing "`()`".



Routine	Arguments	Returns	Function
`compileVarDec`	—	—	Compiles a `var` declaration.
`compileStatements`	—	—	Compiles a sequence of statements, not including the enclosing "{}".
`compileDo`	—	—	Compiles a `do` statement.
`compileLet`	—	—	Compiles a `let` statement.
`compileWhile`	—	—	Compiles a `while` statement.
`compileReturn`	—	—	Compiles a `return` statement.
`compileIf`	—	—	Compiles an `if` statement, possibly with a trailing `else` clause.
`CompileExpression`	—	—	Compiles an expression.
`CompileTerm`	—	—	Compiles a *term*. This routine is faced with a slight difficulty when trying to decide between some of the alternative parsing rules. Specifically, if the current token is an identifier, the routine must distinguish between a variable, an array entry, and a subroutine call. A single look-ahead token, which may be one of "[", "(", or "." suffices to distinguish between the three possibilities. Any other token is not part of this term and should not be advanced over.
`CompileExpressionList`	—	—	Compiles a (possibly empty) comma-separated list of expressions.



THANK YOU