START Final Breakpoint Version Feature Testing

Introduction

This testing document will contain the tests that were carried out as part of the final version of our START debugger, which encompasses both line by line as well as dynamically chosen breakpoints, developed using Java with ANTLR4 for Windows, as part of our START Web Application. This final version builds off of the previous line-by-line version, with some significant changes to how the breakpoints work.

Test Type

Initial Feature Test followed by a Regression Test to fix any bugs initially present in the original implementation.

Test Strategy

The testing strategy will make use of our previously created testing scripts which target the main features of our language, as well as some specific edge cases to attempt to find bugs within our code which without a proper test cycle would not be identified.

If any bugs are found, once the changes have been made to facilitate the fixes needed, we will follow up with a regression test of the initially failed features to validate if they have been fixed.

Feature Identification

1. Variable Assignment
2. Addition
3. Subtraction
4. Multiplication

5. Division
6. Modulus
7. Powers
8. Not Expressions/Parenthesis
9. Comparisons
10. Print Statements
11. Arrays and Array Indexing
12. Boolean Operators
13. If Statements
14. While Loops
15. For Loops
16. Functions
17. Remove All
18. String Indexing
19. Comments
20. Error Generation outputted

Test Data Preparation

The test files used for this process can be found in the project under the following path:

`res/START-test-files/`

Each file will be named accordingly in the Test Data section for traceability of testing. The file of that name can be found under the above path.

Test Execution

Test Name : Variable Assignment

Test Description

Ensure a variable of any type can be assigned and used correctly.

Ensure the breakpoints work as expected.

Test Data

`var-assign.st`

Expected Result

All variables should be shown to be there originally assigned value on output.

There should be (10 points/chosen points) at which the program waits

Memory dynamically updates.

Execution Steps

Code ran in testing version of START Web Application.

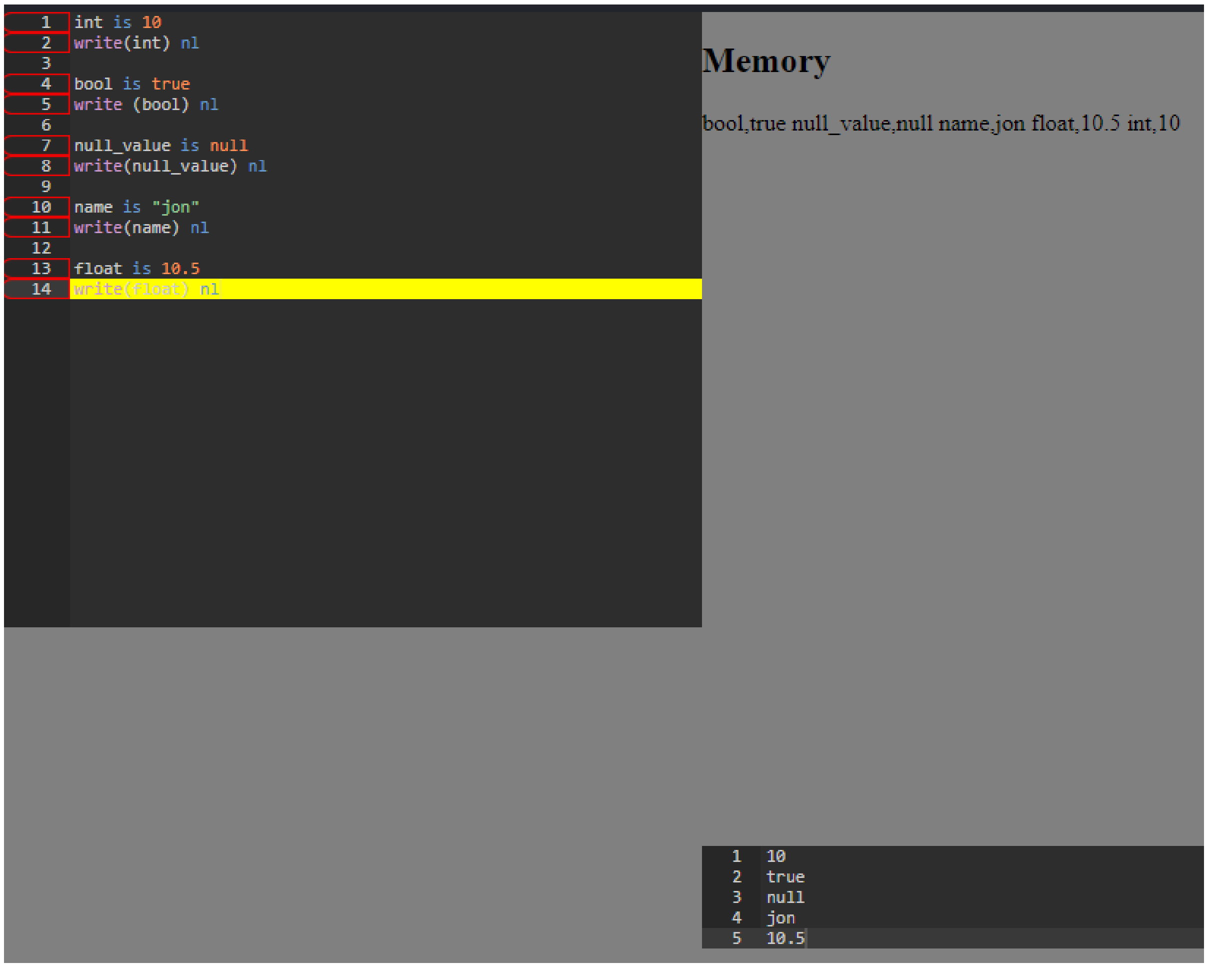
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
  - All variables were of correct value.
  - There was 10 stopping points in the program.
  - Memory updated dynamically.
- Dynamic Mode:
  - All variables were of correct value.
  - The program stopped the correct number of times.
  - Memory updated dynamically.

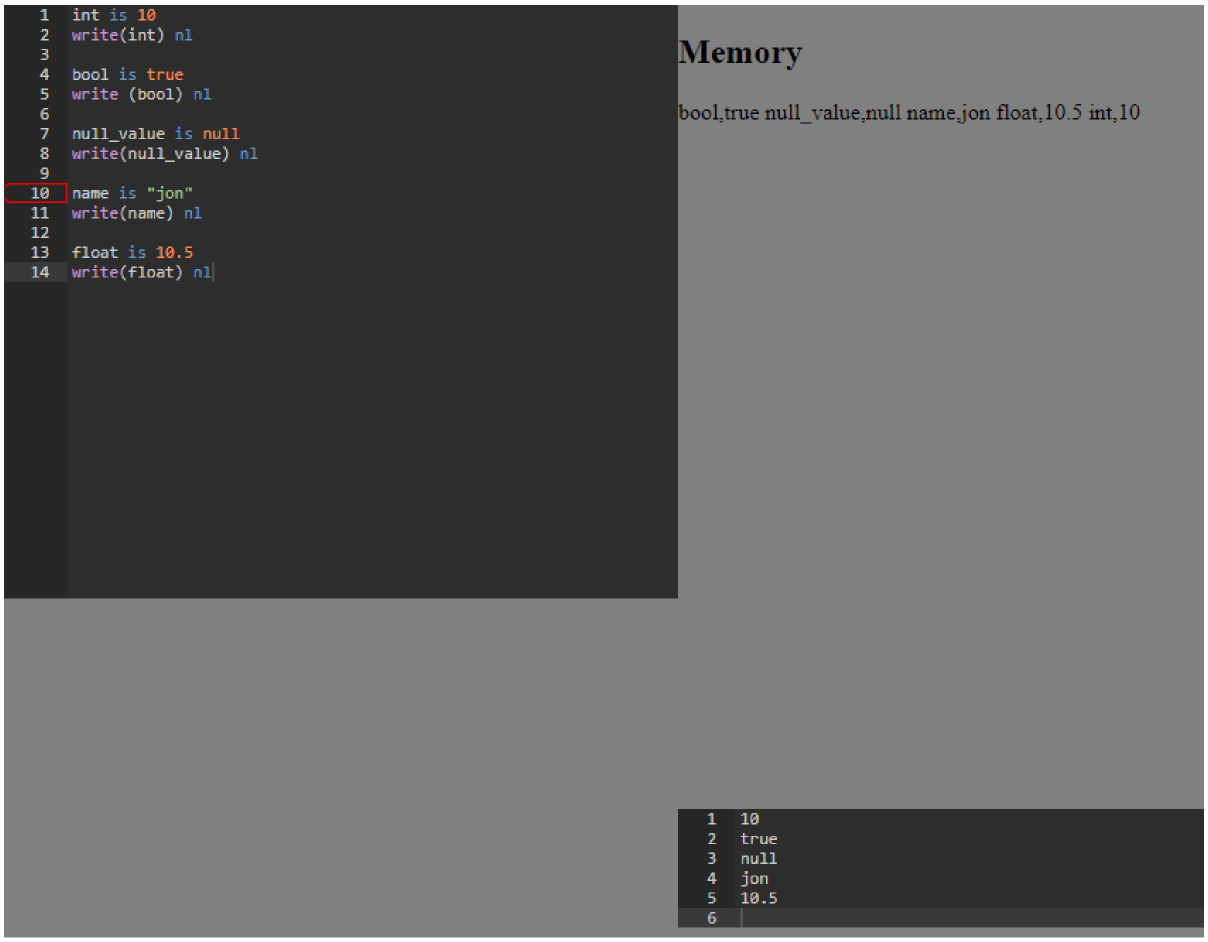
Evidence



```
int is 10
write(int) nl

bool is true
write (bool) nl

null_value is null
write(null_value) nl

name is "jon"
write(name) nl

float is 10.5
write(float) nl
```

bool,true null_value,null name,jon float,10.5 int,10

```
10
true
null
jon
10.5
```



```
int is 10
write(int) nl

bool is true
write (bool) nl

null_value is null
write(null_value) nl

name is "jon"
write(name) nl

float is 10.5
write(float) nl
```

bool,true null_value,null name,jon float,10.5 int,10

```
10
true
null
jon
10.5
```

Overall Result

Pass

Test Name : Addition

Test Description

Ensure assigned variables are able to be added together correctly as well as regular integers and floats within `write()` statements.

Ensure the breakpoints work as expected.

Test Data

`addition.st`

Expected Result

All additions should yield the correct total based on the original variable values provided to the addition operator.

There should be (10 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

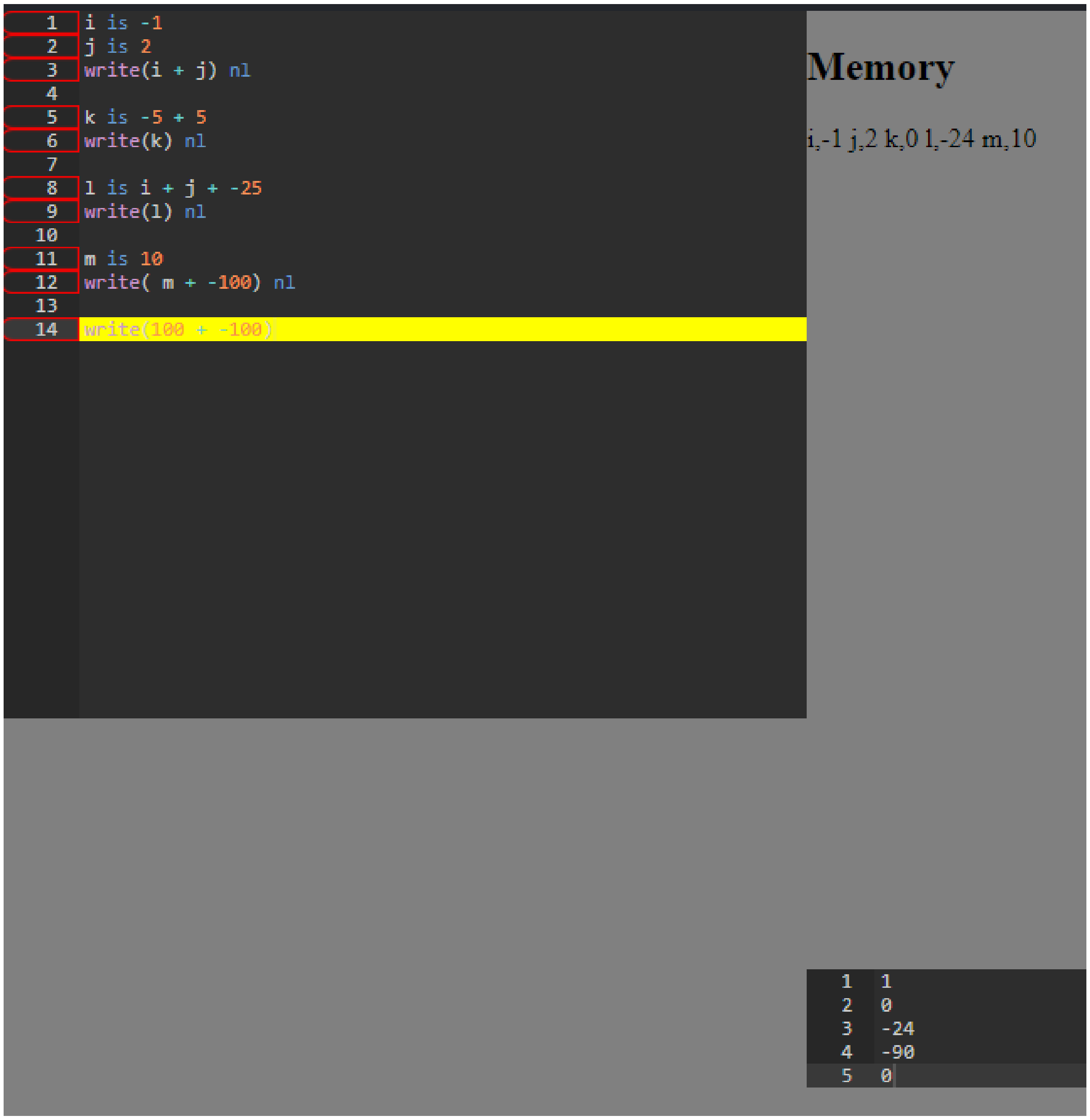
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
    - All additions were correct.
    - There was 10 stopping points in the program.
    - Memory updated dynamically.
- Dynamic Mode:
    - All additions were correct.
    - The program stopped the correct number of times.
    - Memory updated dynamically.

Evidence

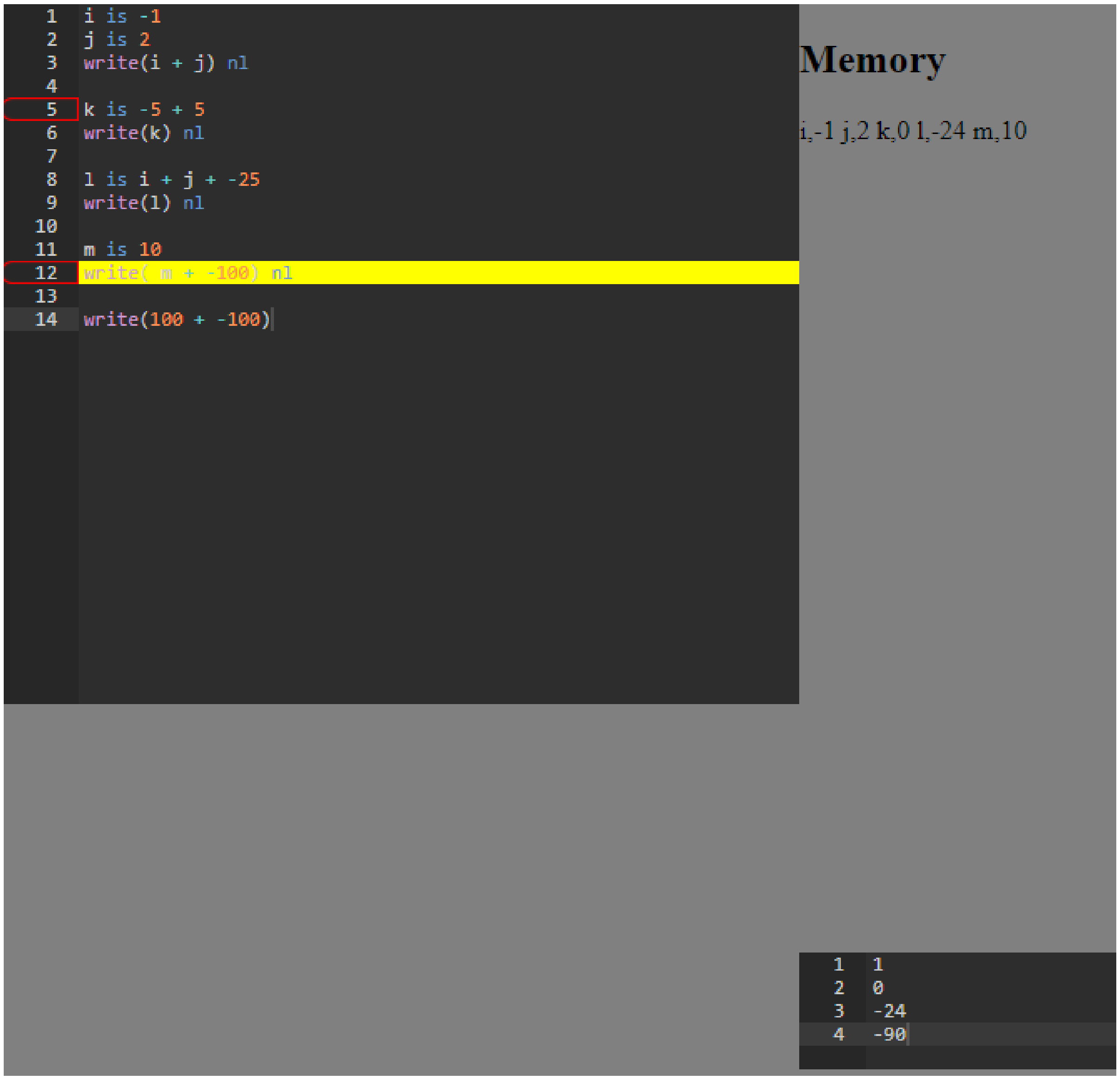


```
i is -1
j is 2
write(i + j) nl

k is -5 + 5
write(k) nl

l is i + j + -25
write(l) nl

m is 10
write( m + -100) nl

write(100 + -100)
```

Memory

i,-1 j,2 k,0 l,-24 m,10

```
1
0
-24
-90
0
```



```
i is -1
j is 2
write(i + j) nl

k is -5 + 5
write(k) nl

l is i + j + -25
write(l) nl

m is 10
write( m + -100) nl

write(100 + -100)
```

Memory

i,-1 j,2 k,0 l,-24 m,10

```
1
0
-24
-90
```

Overall Result

Pass

Test Name : Subtraction

Test Description

Ensure assigned variables are able to be subtracted from each other correctly as well as regular integers and floats within `write()` statements.

Ensure the breakpoints work as expected.

Test Data

`subtraction.st`

Expected Result

All subtractions should yield the correct total based on the original variable values provided to the subtraction operator.

There should be (10 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

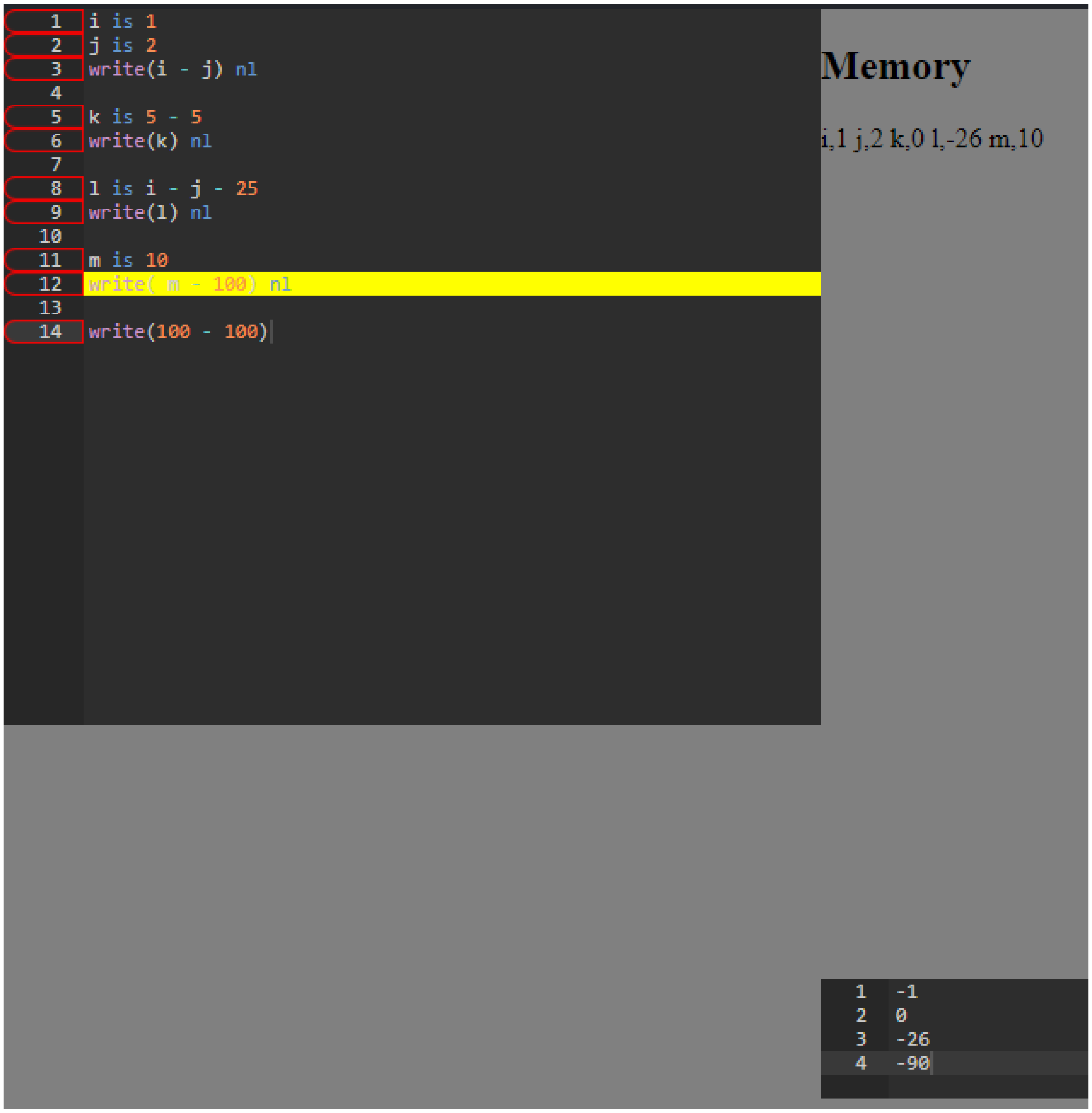
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
    - All subtractions were correct.
    - There was 10 stopping points in the program.
    - Memory updated dynamically.
- Dynamic Mode:
    - All subtractions were correct.
    - The program stopped the correct number of times.
    - Memory updated dynamically.

Evidence

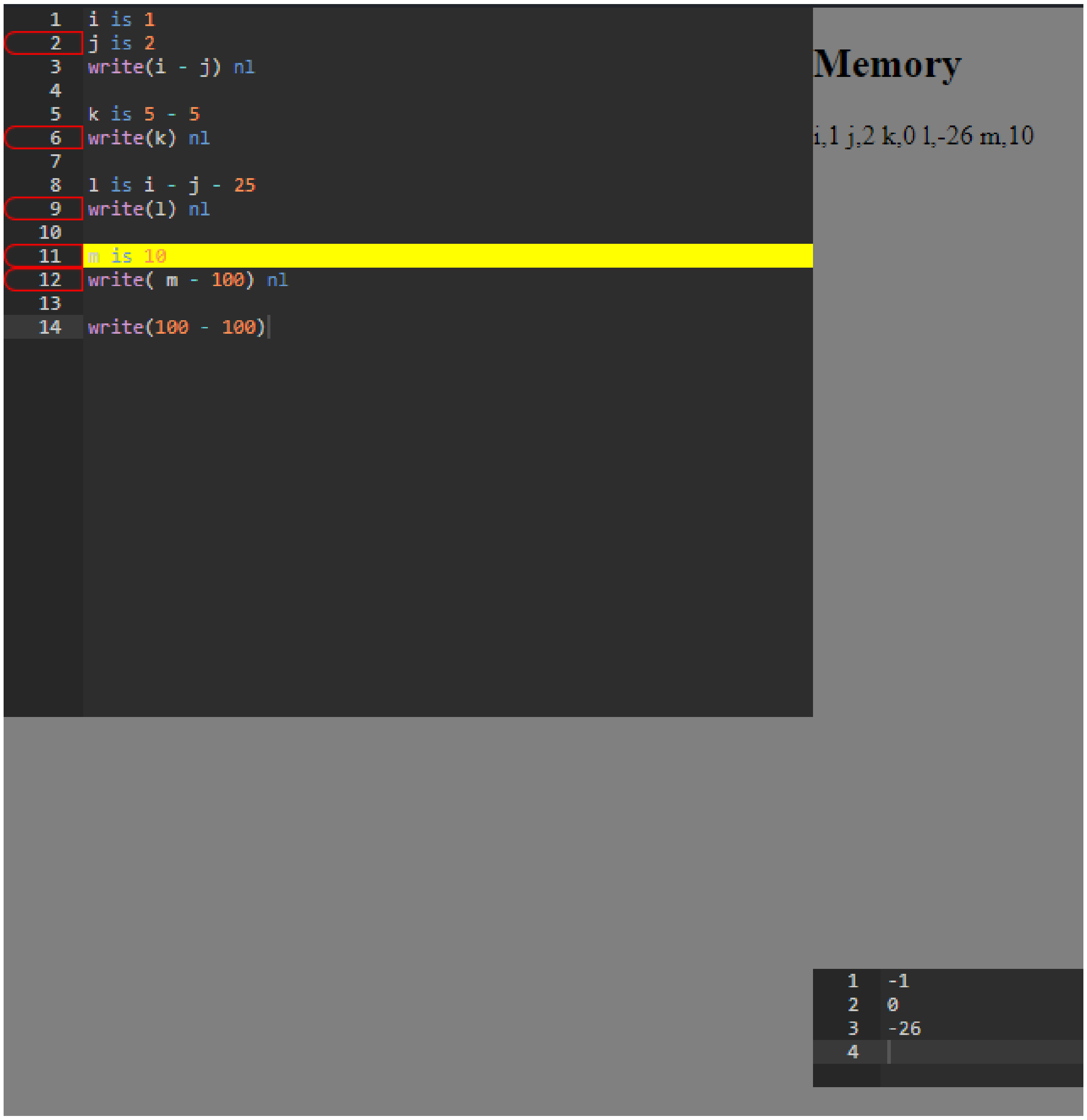


```
i is 1
j is 2
write(i - j) nl

k is 5 - 5
write(k) nl

l is i - j - 25
write(l) nl

m is 10
write( m - 100) nl

write(100 - 100)
```

Memory

i,1 j,2 k,0 l,-26 m,10

```
-1
0
-26
-90
```



```
i is 1
j is 2
write(i - j) nl

k is 5 - 5
write(k) nl

l is i - j - 25
write(l) nl

m is 10
write( m - 100) nl

write(100 - 100)
```

Memory

i,1 j,2 k,0 l,-26 m,10

```
-1
0
-26
```

Overall Result

Pass

Test Name : Multiplication

Test Description

Ensure assigned variables are able to be multiplied together correctly as well as regular integers and floats within `write()` statements.

Ensure the breakpoints work as expected.

Test Data

`multiplication.st`

Expected Result

All multiplications should yield the correct total based on the original variable values provided to the multiplication operator.

There should be (13 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

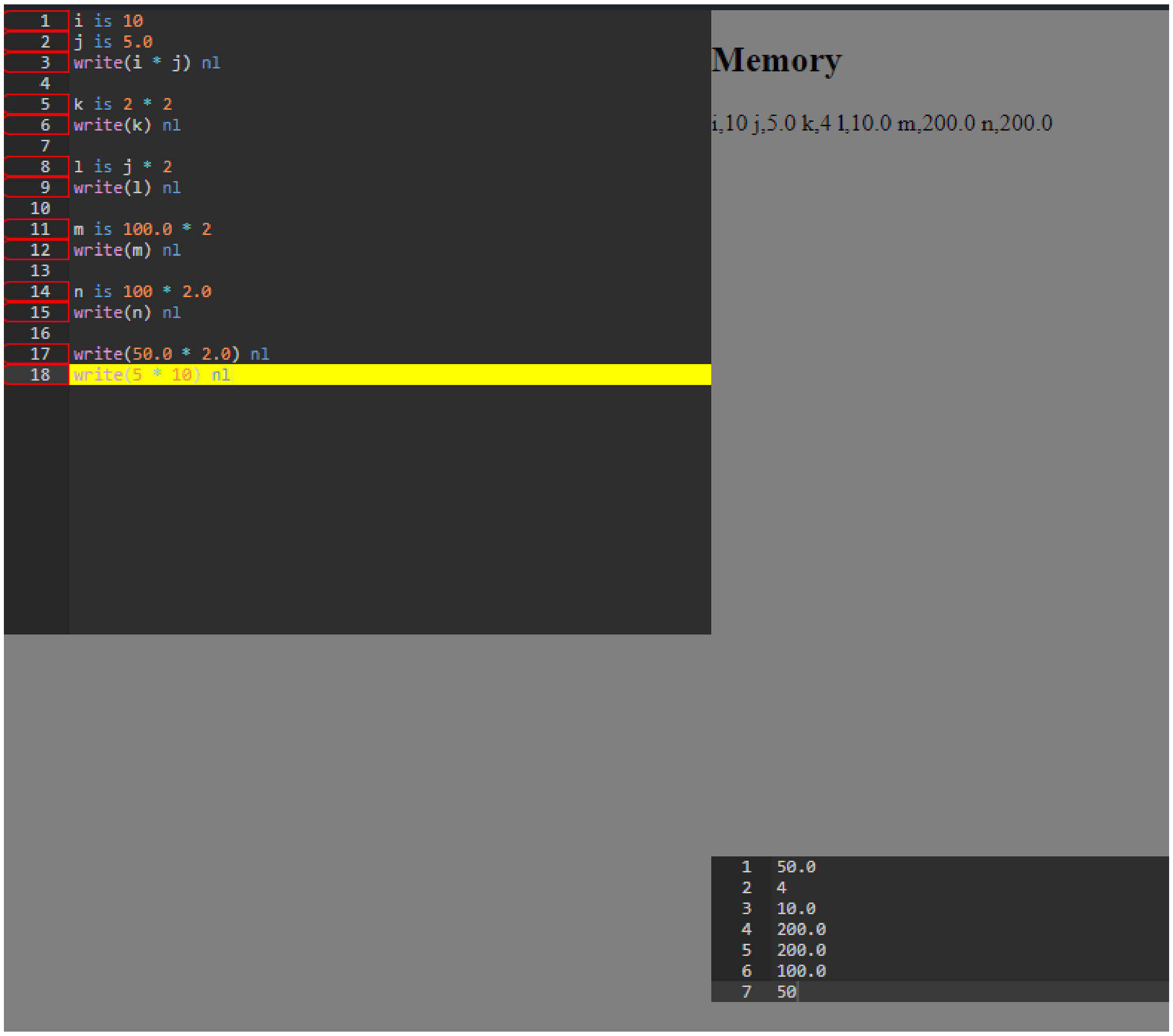
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
    - All multiplications were correct.
    - There was 13 stopping points in the program.
    - Memory updated dynamically.
- Dynamic Mode:
    - All multiplications were correct.
    - The program stopped the correct number of times.
    - Memory updated dynamically.

Evidence

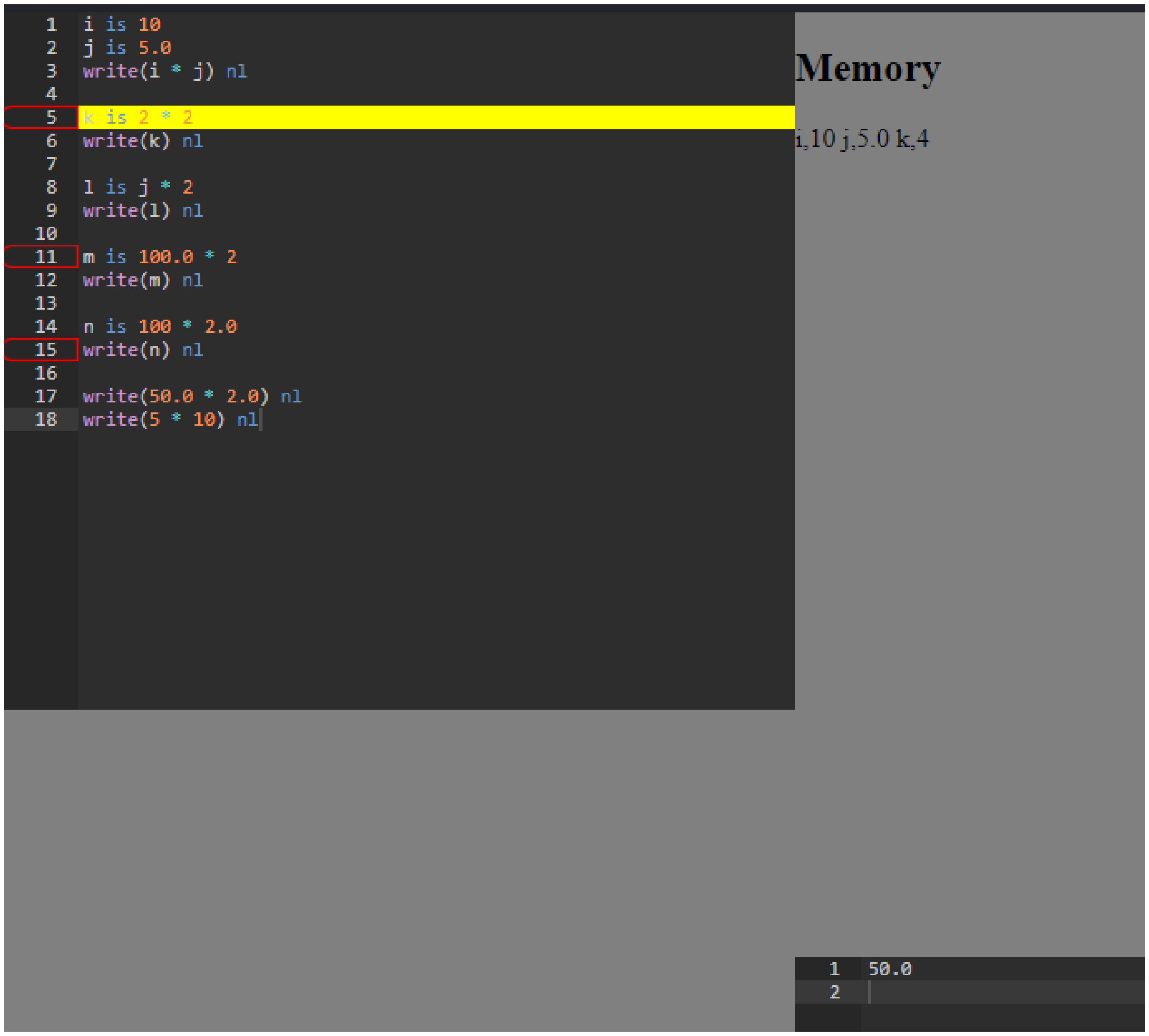


```
i is 10
j is 5.0
write(i * j) nl

k is 2 * 2
write(k) nl

l is j * 2
write(l) nl

m is 100.0 * 2
write(m) nl

n is 100 * 2.0
write(n) nl

write(50.0 * 2.0) nl
write(5 * 10) nl
```

Memory

i,10 j,5.0 k,4 l,10.0 m,200.0 n,200.0

```
50.0
4
10.0
200.0
200.0
100.0
50
```



```
i is 10
j is 5.0
write(i * j) nl

k is 2 * 2
write(k) nl

l is j * 2
write(l) nl

m is 100.0 * 2
write(m) nl

n is 100 * 2.0
write(n) nl

write(50.0 * 2.0) nl
write(5 * 10) nl
```

Memory

i,10 j,5.0 k,4

```
50.0
```

Overall Result

Pass

Test Name : Division

Test Description

Ensure two assigned variables are divisible as well as regular integers and floats within `write()` statements.

Ensure the breakpoints work as expected.

Test Data

`division.st`

Expected Result

All sets of division should yield the correct total based on the original variable values provided to the division operator.

There should be (12 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

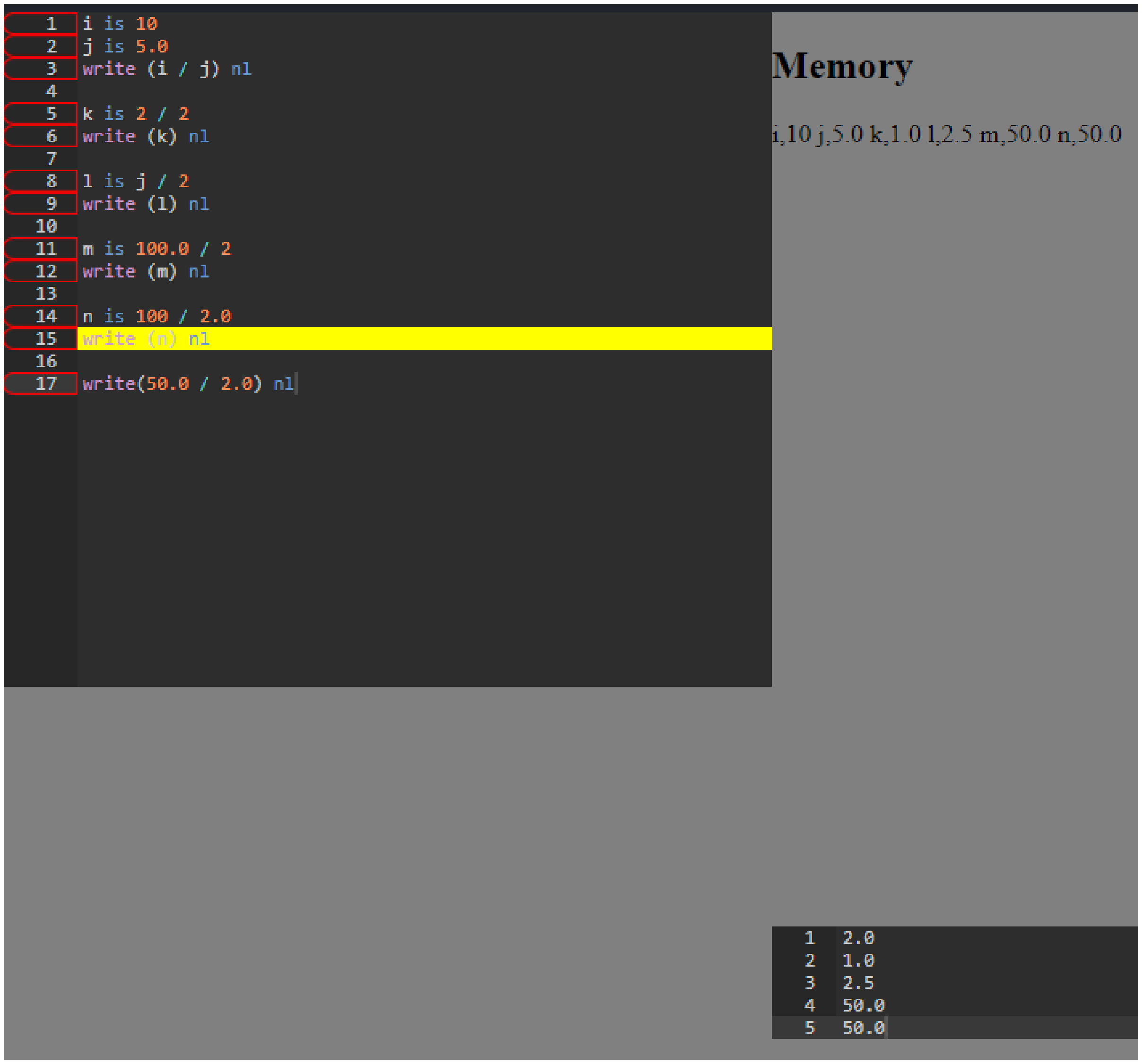
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
    - All divisions were correct.
    - There was 12 stopping points in the program.
    - Memory updated dynamically.
- Dynamic Mode:
    - All divisions were correct.
    - The program stopped the correct number of times.
    - Memory updated dynamically.

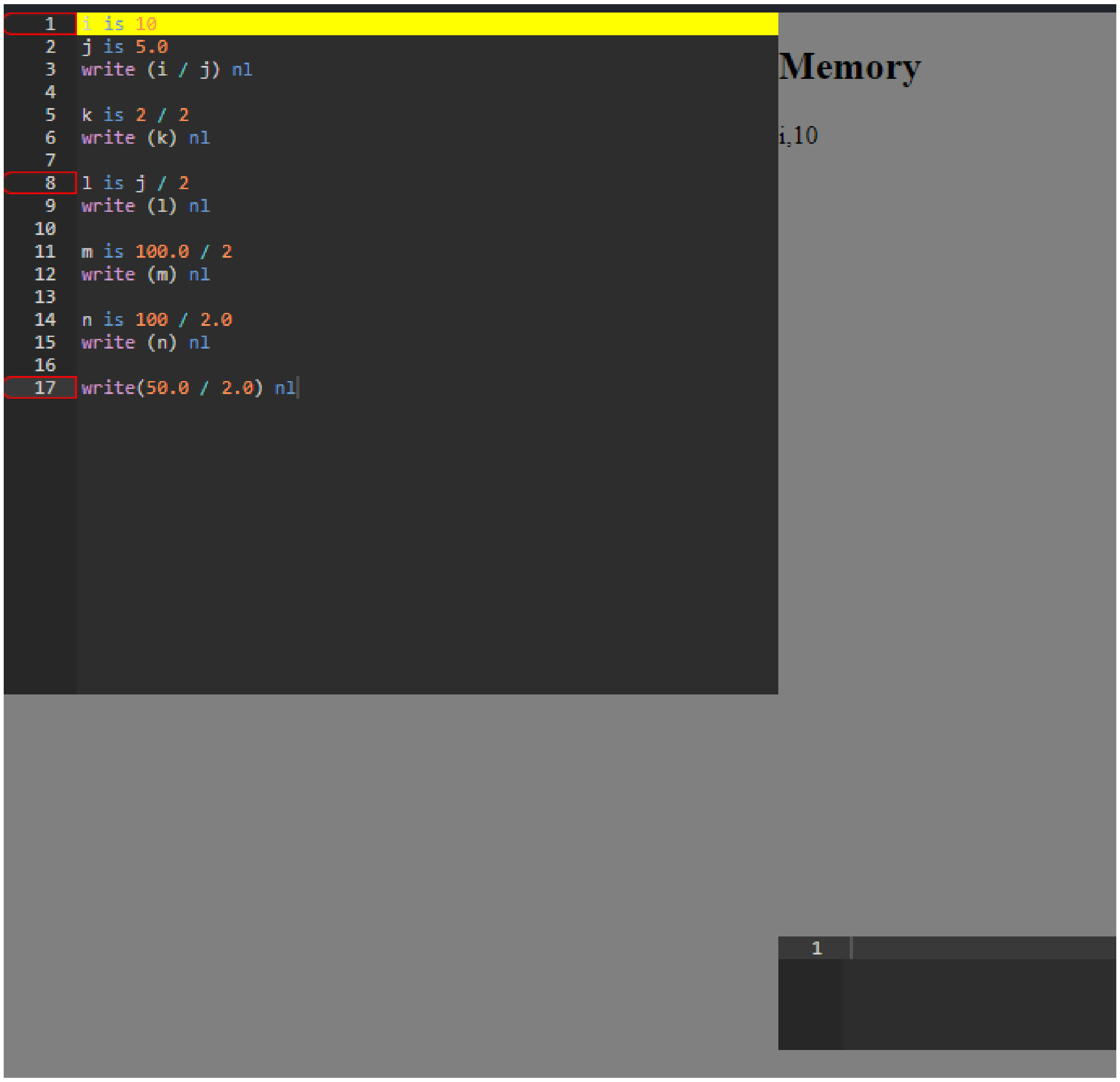
Evidence



```
i is 10
j is 5.0
write (i / j) nl

k is 2 / 2
write (k) nl

l is j / 2
write (l) nl

m is 100.0 / 2
write (m) nl

n is 100 / 2.0
write (n) nl

write(50.0 / 2.0) nl
```

i,10 j,5.0 k,1.0 l,2.5 m,50.0 n,50.0

```
2.0
1.0
2.5
50.0
50.0
```



```
i is 10
j is 5.0
write (i / j) nl

k is 2 / 2
write (k) nl

l is j / 2
write (l) nl

m is 100.0 / 2
write (m) nl

n is 100 / 2.0
write (n) nl

write(50.0 / 2.0) nl
```

Overall Result

Pass

Test Name : Modulus

Test Description

All modulus operations should yield the correct remainder after one variable is divided by another, as well as any integer or float within a `write()` statement.

Ensure the breakpoints work as expected.

Test Data

`modulus.st`

Expected Result

The correct remained after division is shown for each modulus operation.

There should be (9 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

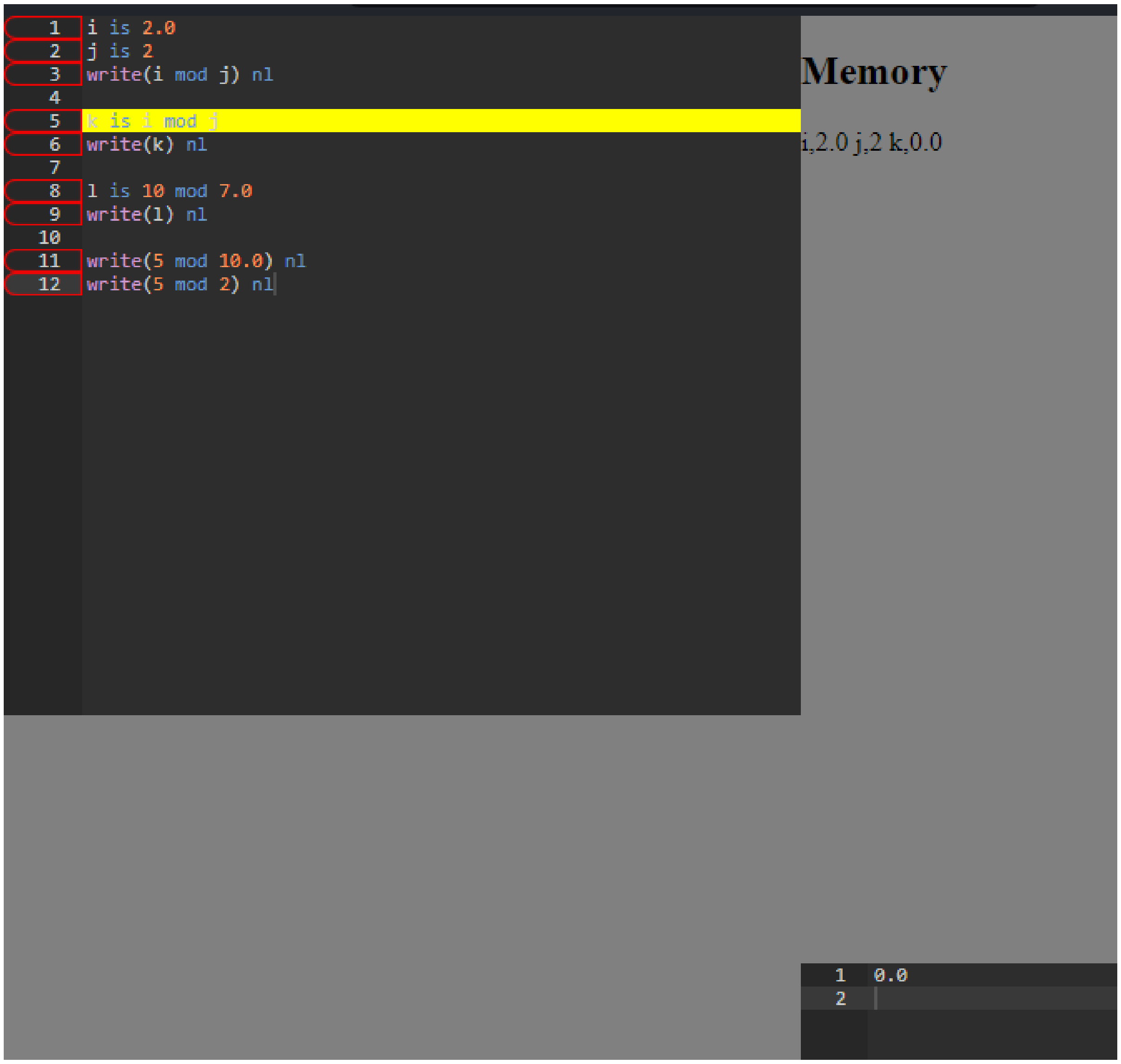
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
    - All remainders were correct.
    - There was 9 stopping points in the program.
    - Memory updated dynamically.
- Dynamic Mode:
    - All remainders were correct.
    - The program stopped the correct number of times.
    - Memory updated dynamically.

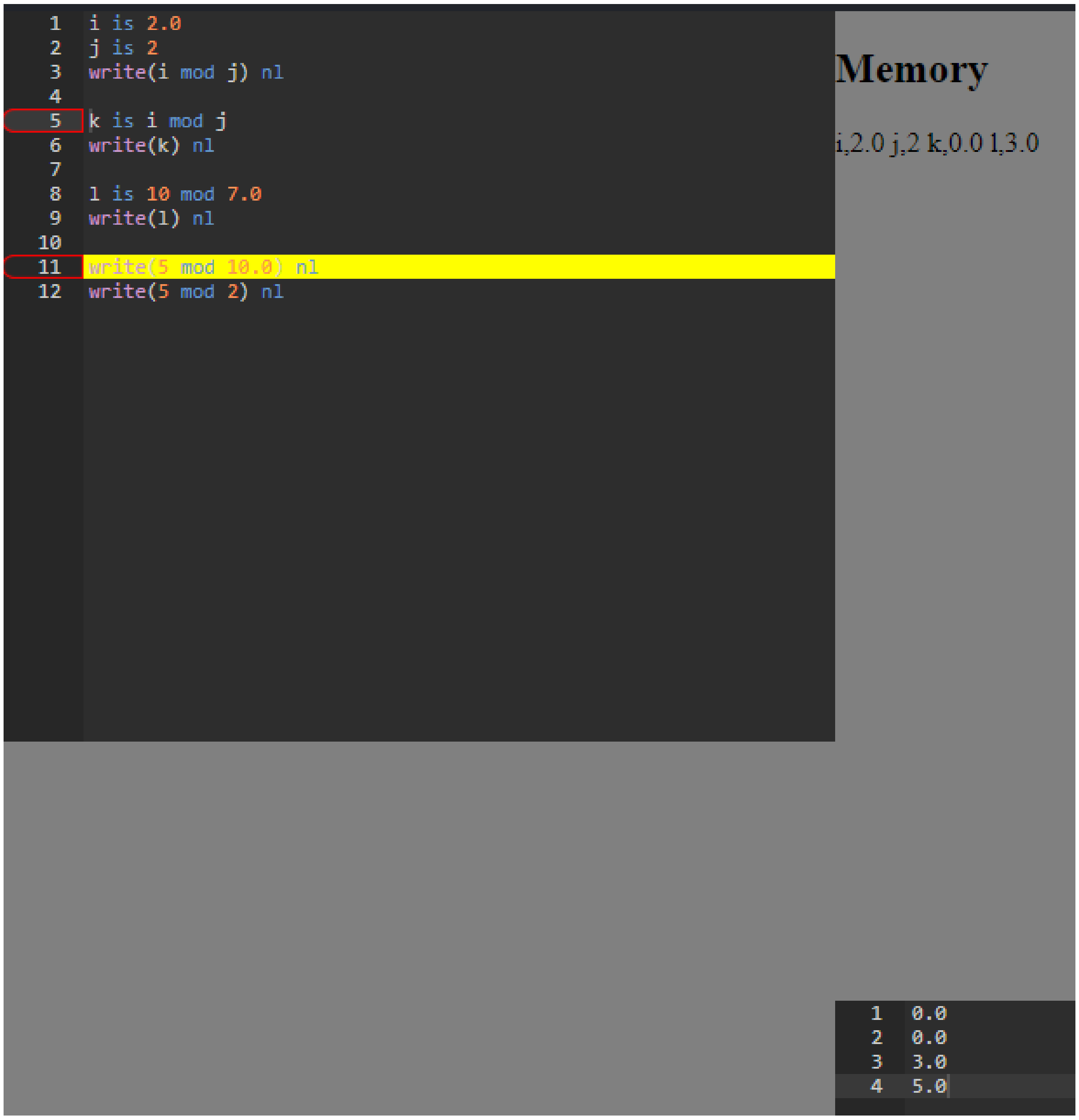
Evidence



```
i is 2.0
j is 2
write(i mod j) nl

k is i mod j
write(k) nl

l is 10 mod 7.0
write(l) nl

write(5 mod 10.0) nl
write(5 mod 2) nl
```

```
i,2.0 j,2 k,0.0
```

```
0.0
```



```
i is 2.0
j is 2
write(i mod j) nl

k is i mod j
write(k) nl

l is 10 mod 7.0
write(l) nl

write(5 mod 10.0) nl
write(5 mod 2) nl
```

```
i,2.0 j,2 k,0.0 l,3.0
```

```
1  0.0
2  0.0
3  3.0
4  5.0
```

Overall Result

Pass

Test Name : Powers

Test Description

The correct result should be produced when one variable is put to the power of another, as well as two integers or floats within a `write()` statement.

Ensure the breakpoints work as expected.

Test Data

`powers.st`

Expected Result

All results are the expected value when a power operation is performed.

There should be (8 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

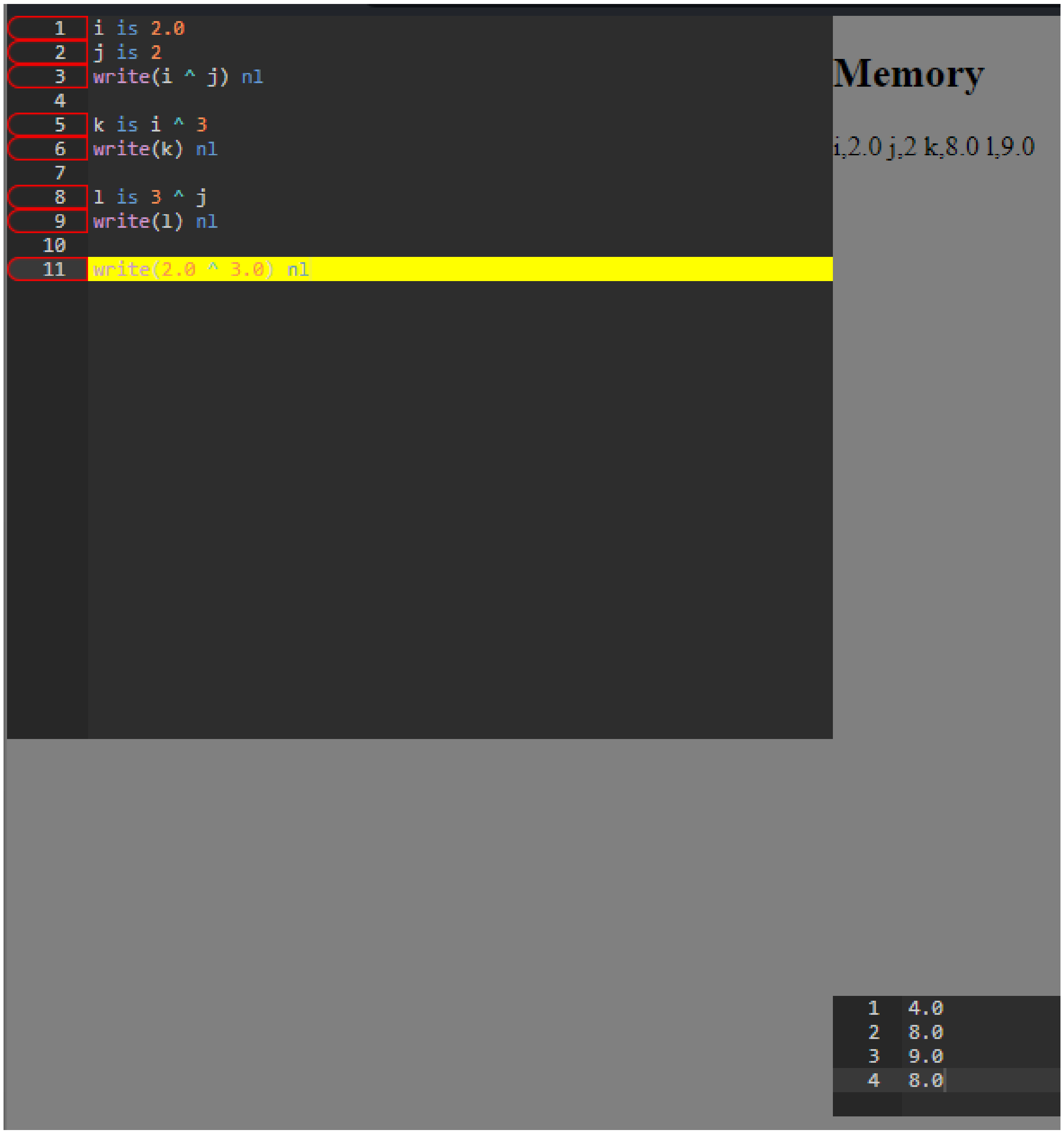
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
  - All results were of correct value.
  - There was 8 stopping points in the program.
  - Memory updated dynamically.
- Dynamic Mode:
  - All results were of correct value.
  - The program stopped the correct number of times.
  - Memory updated dynamically.

Evidence

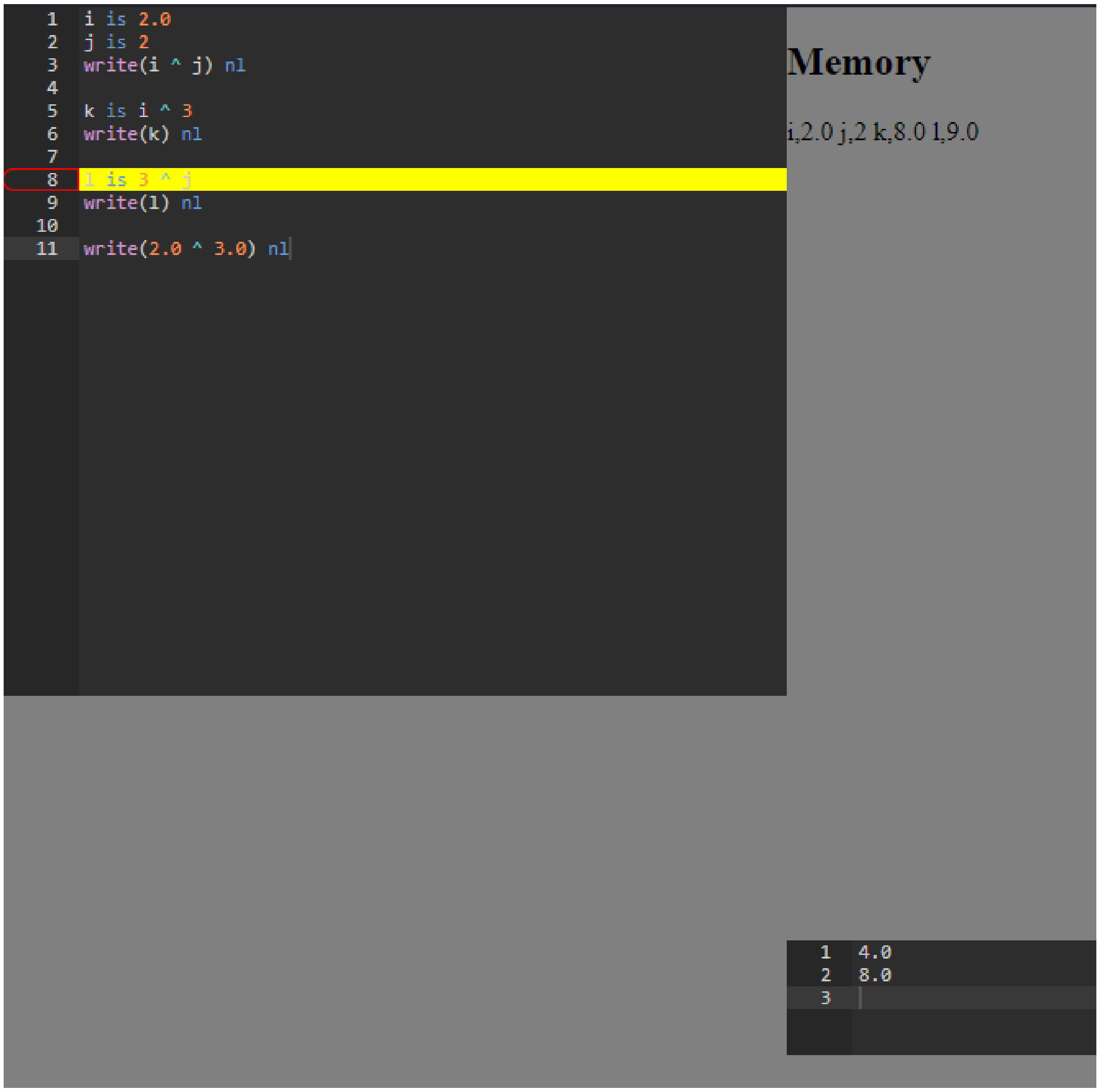


```
i is 2.0
j is 2
write(i ^ j) nl

k is i ^ 3
write(k) nl

l is 3 ^ j
write(l) nl

write(2.0 ^ 3.0) nl
```

Memory

i,2.0 j,2 k,8.0 l,9.0

```
1   4.0
2   8.0
3   9.0
4   8.0
```



```
i is 2.0
j is 2
write(i ^ j) nl

k is i ^ 3
write(k) nl

l is 3 ^ j
write(l) nl

write(2.0 ^ 3.0) nl
```

```
i,2.0 j,2 k,8.0 l,9.0
```

```
4.0
8.0
```

Overall Result

Pass

Test Name : Not Expressions & Parenthesis

Test Description

The logical not operator should be able to swap the result of a boolean value that is calculated. As well as this we need to ensure the parenthesis take precedence in the operations.

Ensure the breakpoints work as expected.

Test Data

`not-paren.st`

Expected Result

The final values returned are the opposite of what the operation inside of the parenthesis states, hence the logical negation would work.

There should be (7 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

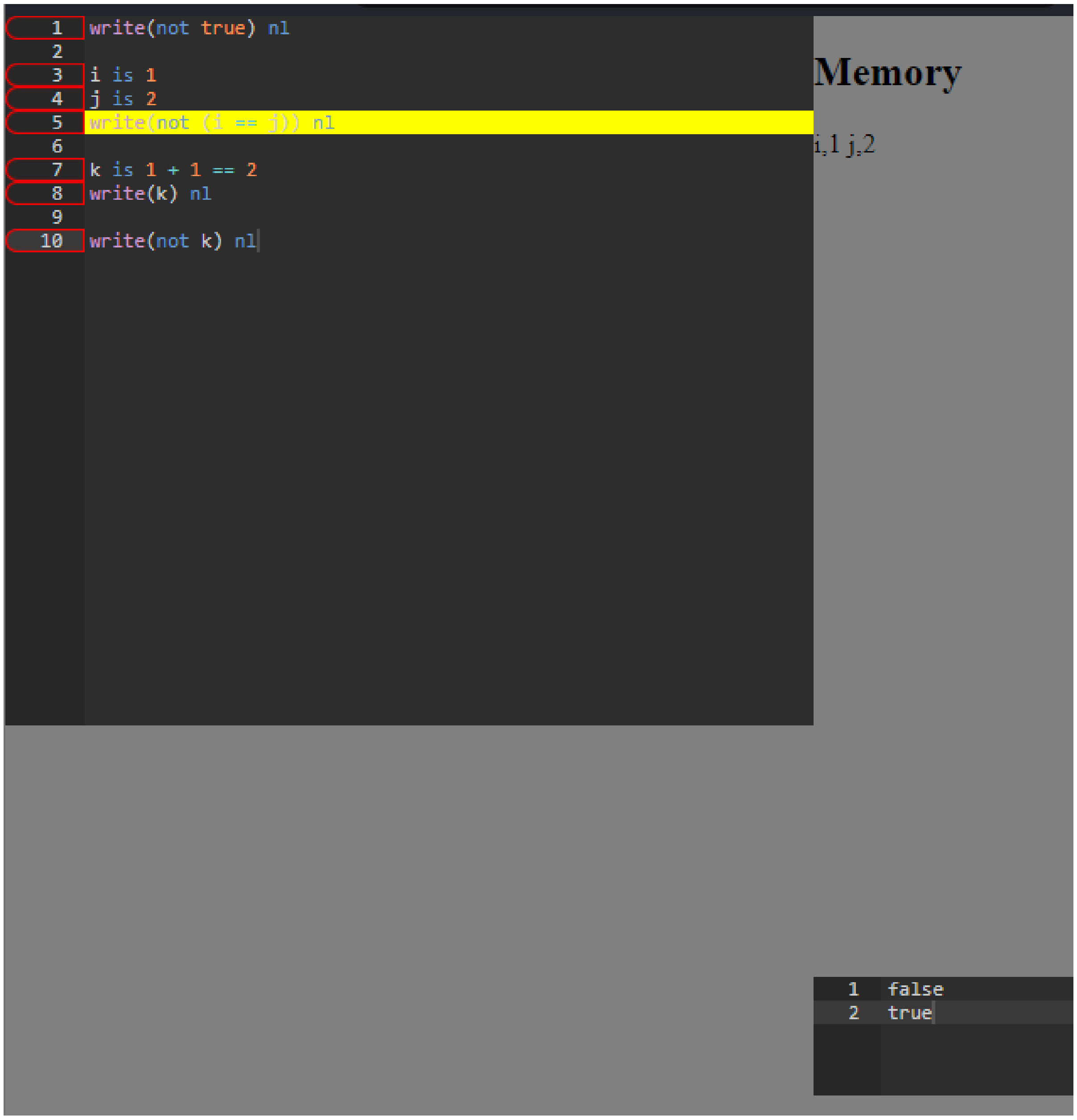
Step button used to advance the program past it’s breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
  - All values were of correct value.
  - There was 7 stopping points in the program.
  - Memory updated dynamically.
- Dynamic Mode:
  - All values were of correct value.
  - The program stopped the correct number of times.
  - Memory updated dynamically.

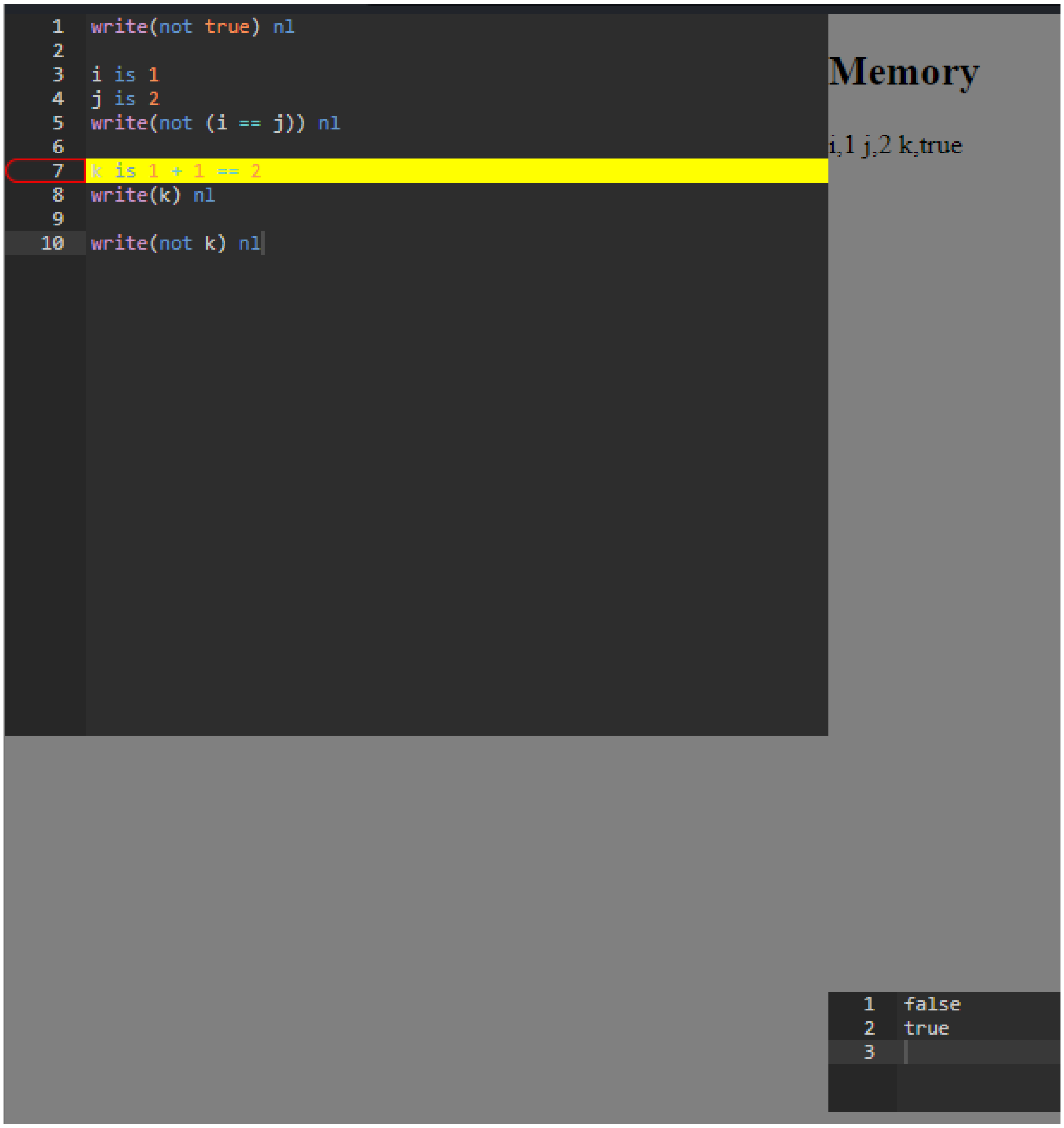
Evidence



```
write(not true) nl

i is 1
j is 2
write(not (i == j)) nl

k is 1 + 1 == 2
write(k) nl

write(not k) nl
```

i,1 j,2

```
false
true
```



```
write(not true) nl

i is 1
j is 2
write(not (i == j)) nl

k is 1 + 1 == 2
write(k) nl

write(not k) nl
```

Memory

i,1 j,2 k,true

```
false
true
```

Overall Result

Pass

Test Name : Comparisons

Test Description

All comparison operators should be able to check if the given condition evaluates to either true or false.

Ensure the breakpoints work as expected.

Test Data

`comp.st`

Expected Result

We expect to see all comparisons yield the correct boolean value once the comparison has taken place and has been written out.

There should be (12 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

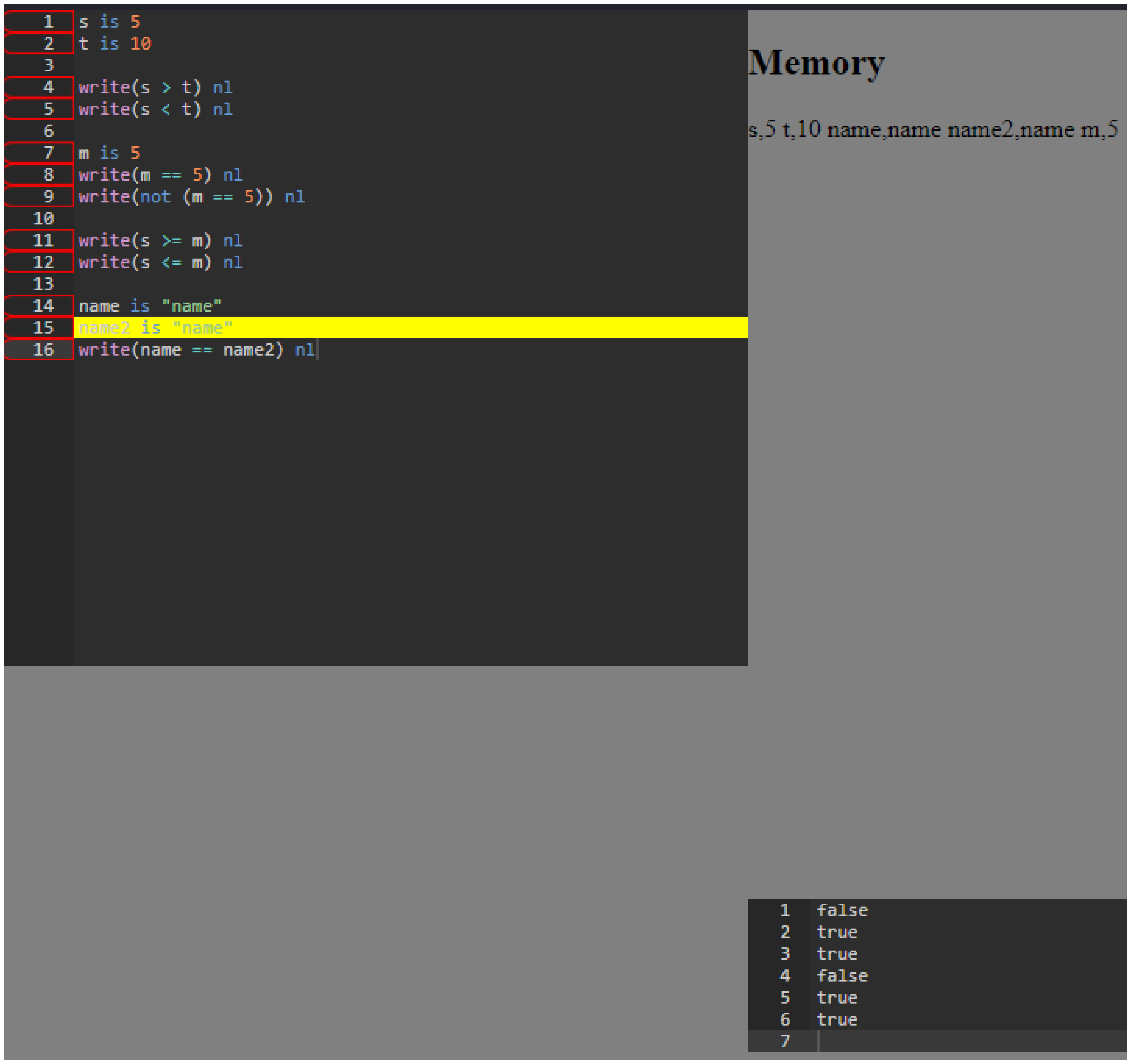
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
  - All boolean values were of correct value.
  - There was 12 stopping points in the program.
  - Memory updated dynamically.
- Dynamic Mode:
  - All boolean values were of correct value.
  - The program stopped the correct number of times.
  - Memory updated dynamically.

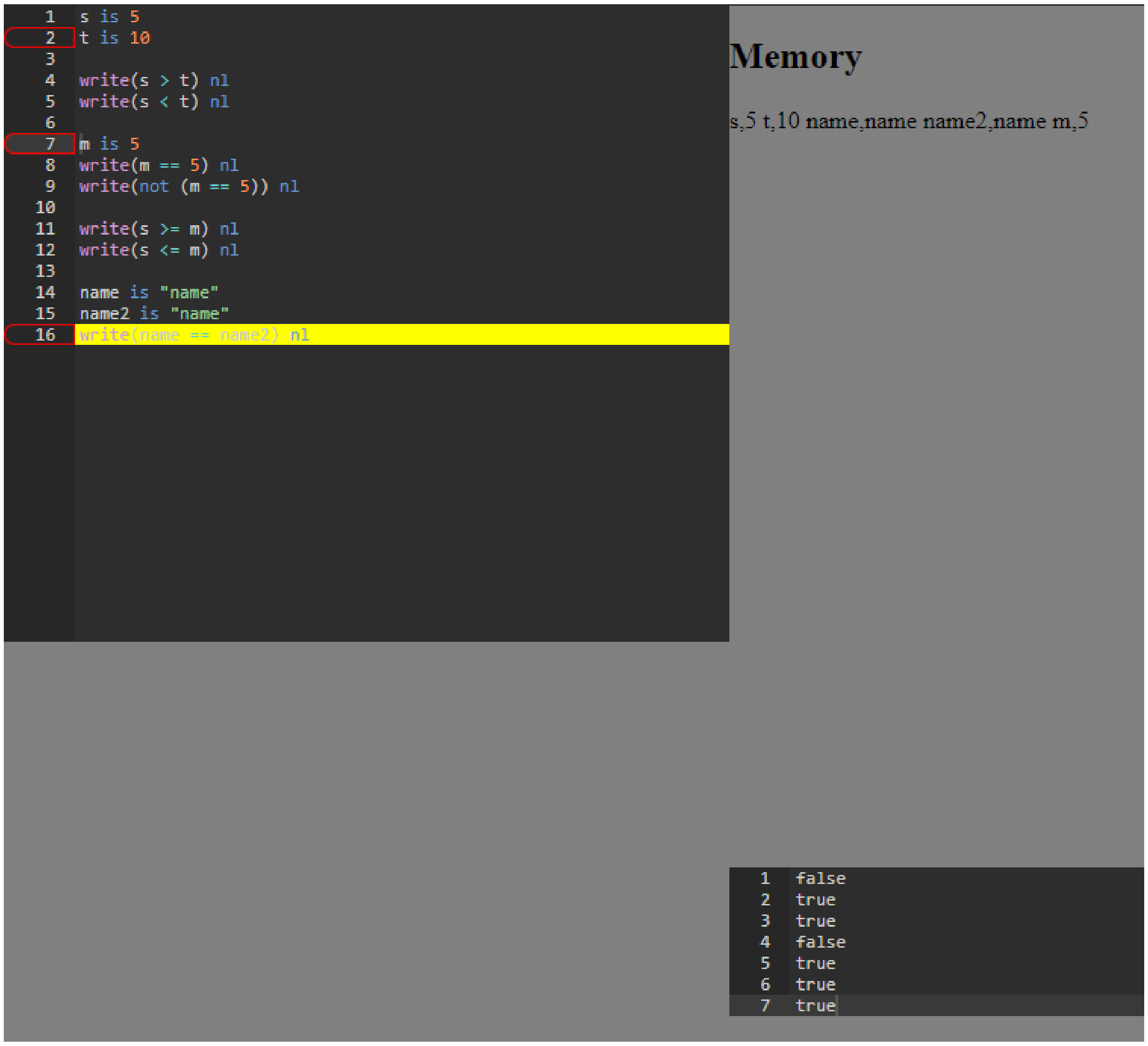
Evidence



```
s is 5
t is 10

write(s > t) nl
write(s < t) nl

m is 5
write(m == 5) nl
write(not (m == 5)) nl

write(s >= m) nl
write(s <= m) nl

name is "name"
name2 is "name"
write(name == name2) nl
```

s,5 t,10 name,name name2,name m,5

```
false
true
true
false
true
true
```



```
s is 5
t is 10

write(s > t) nl
write(s < t) nl

m is 5
write(m == 5) nl
write(not (m == 5)) nl

write(s >= m) nl
write(s <= m) nl

name is "name"
name2 is "name"
write(name == name2) nl
```

s,5 t,10 name,name name2,name m,5

```
false
true
true
false
true
true
true
```

Overall Result

Pass

Test Name : Print Statements

Test Description

The built in `write()` function must be able to output to the console, both with and without a newline.

Ensure the breakpoints work as expected.

Test Data

`print.st`

Expected Result

We expect to see the output of the `write()` function.

There should be (4 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

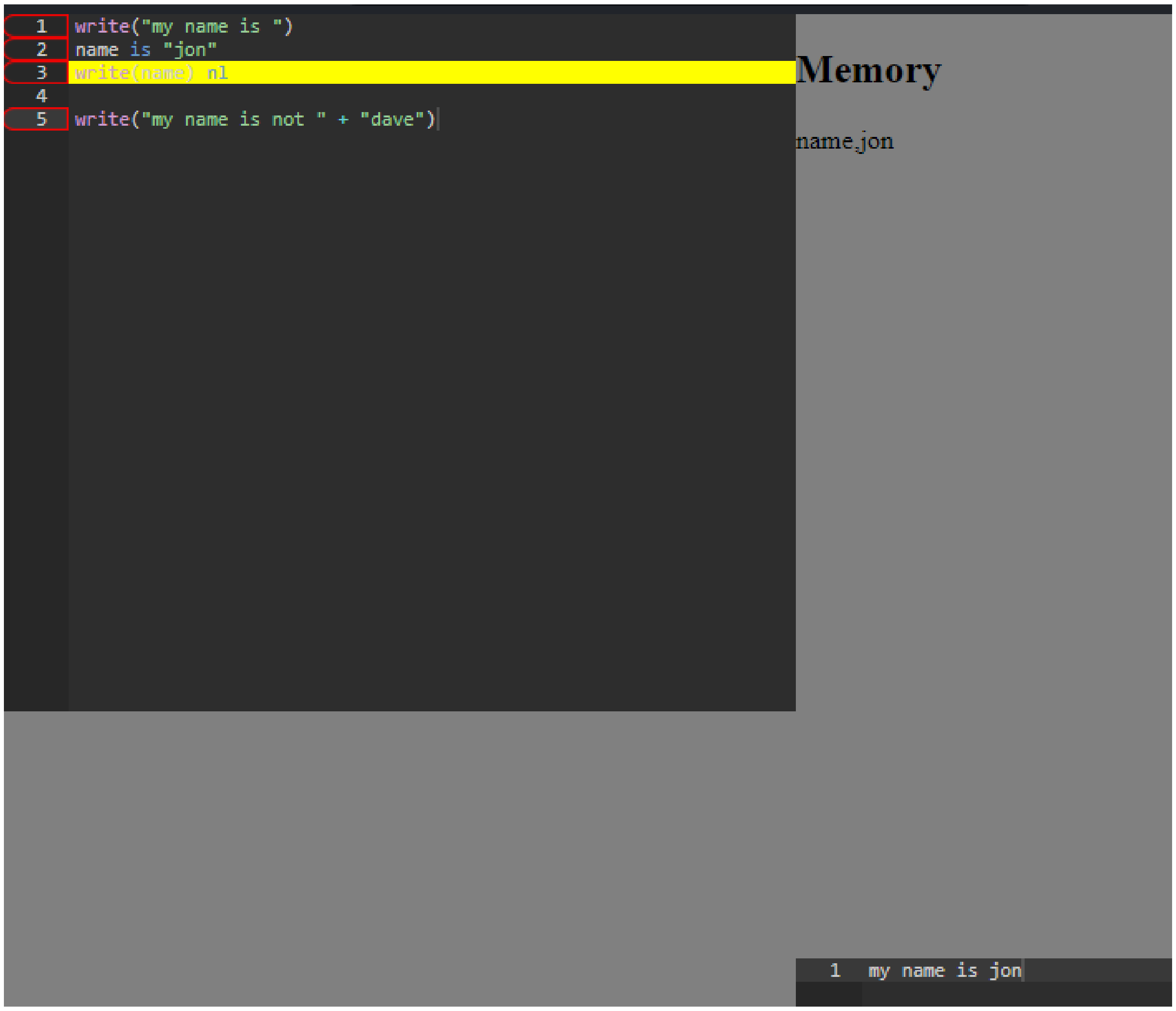
Step button used to advance the program past it’s breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
    - Output was seen.
    - There was 4 stopping points in the program.
    - Memory updated dynamically.
- Dynamic Mode:
    - Output was seen.
    - The program stopped the correct number of times.
    - Memory updated dynamically.

Evidence

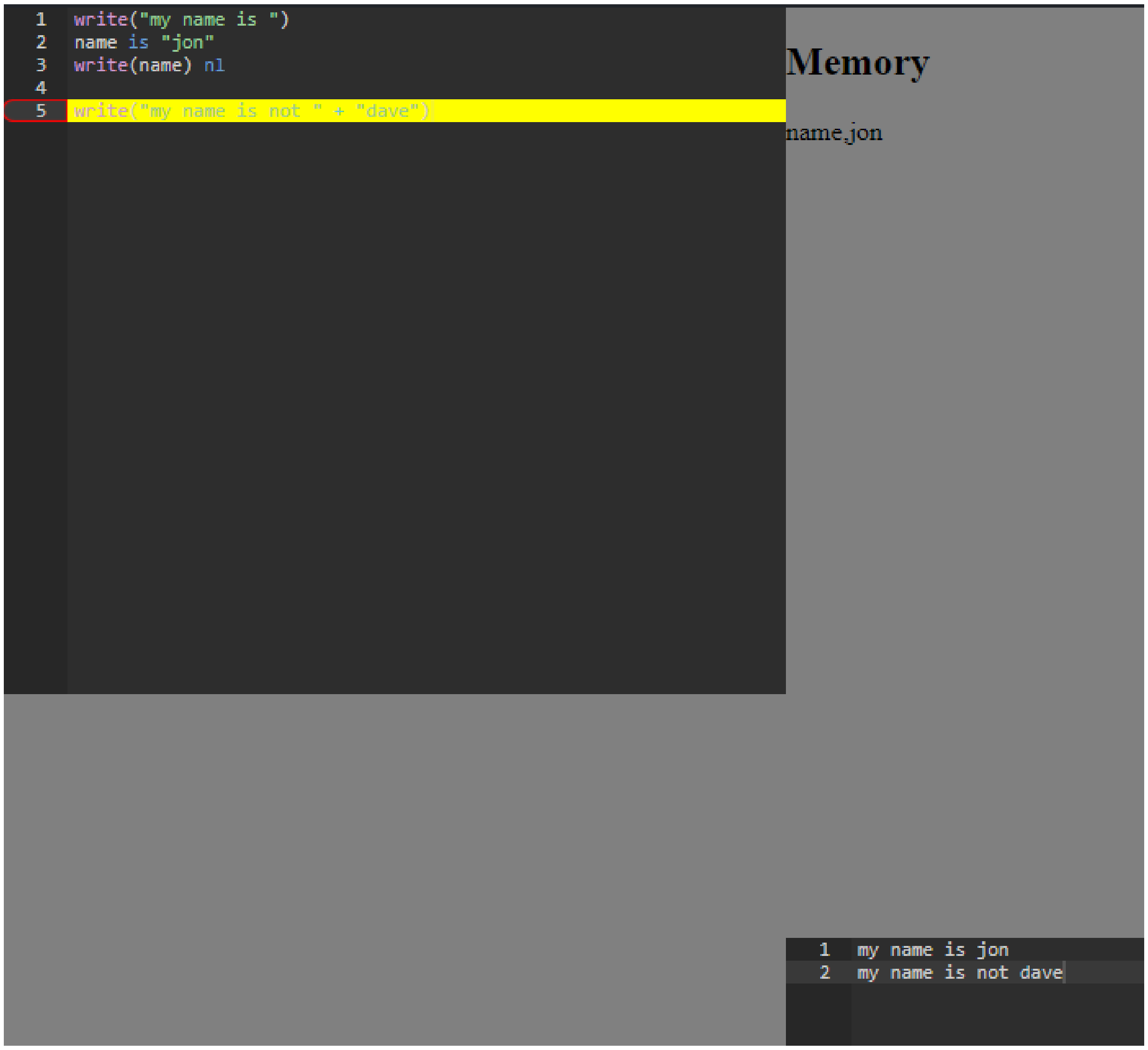


```
write("my name is ")
name is "jon"
write(name) nl

write("my name is not " + "dave")
```

Memory

name.jon

```
1  my name is jon
```



```
write("my name is ")
name is "jon"
write(name) nl

write("my name is not " + "dave")
```

```
name,jon
```

```
my name is jon
my name is not dave
```

Overall Result

Pass

Test Name : Arrays & Array Indexing

Test Description

Arrays must be able to support the following features:

- Printing
- Multi type content
- Indexing the array to print

- Changing values at an index
- Appending to the end of an array
- Concatenation of 2 arrays
- Getting the length of the array
- Removing from an array

Ensure the breakpoints work as expected.

Test Data

`arr-arrIndex.st`

`arr-append.st`

`arr-concat.st`

`arr-length.st`

`arr-remove.st`

Expected Result

- An array is printed
- Arrays contain multi content successfully
- Array index retrieved and changed
- 2 arrays are concatenated
- The length is found of an array
- Items are removed from an array

The correct number of breakpoints are present in each file.

Execution Steps

Code ran in testing version of START Web Application.

Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

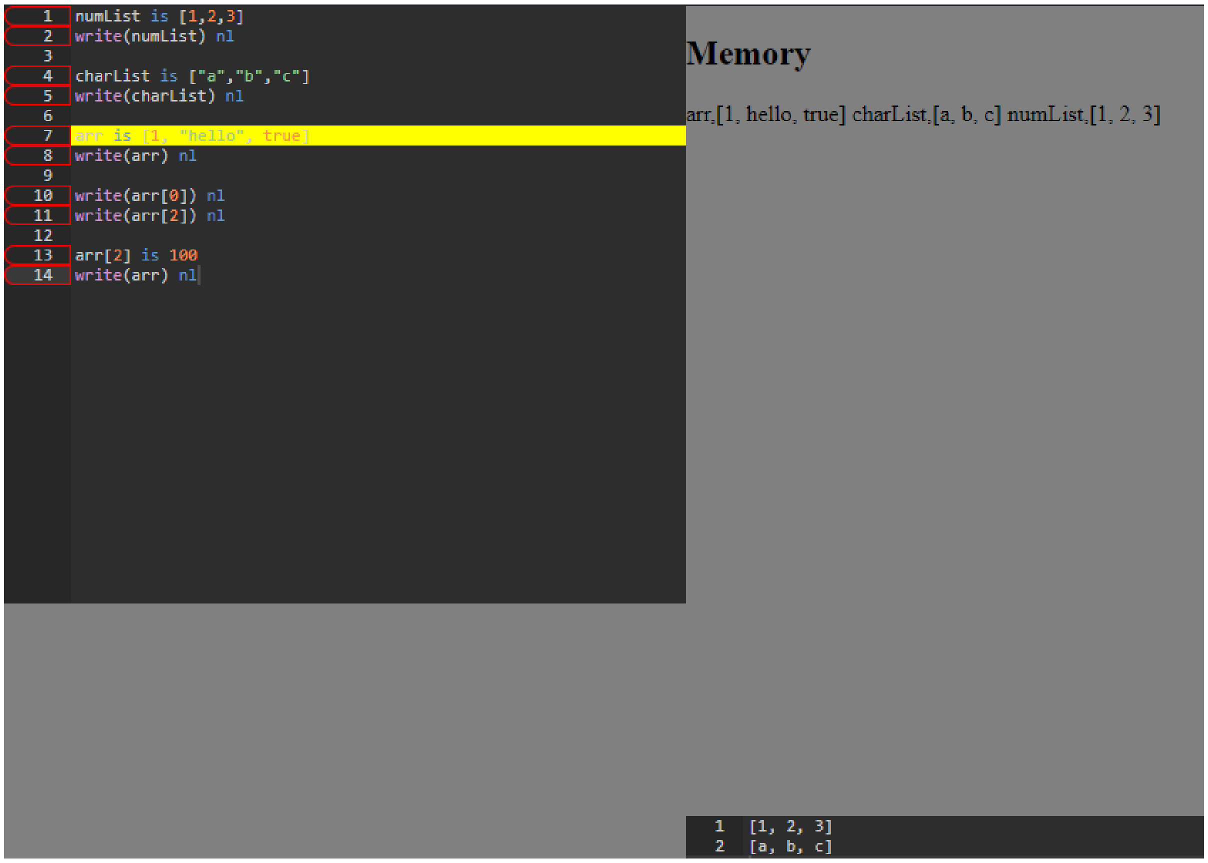
Actual Result

All cases were passed, and all expected results were found to be correct, meaning the arrays are fully functional.

The correct number of breakpoints where present per file.

The memory dynamically updated.

Evidence

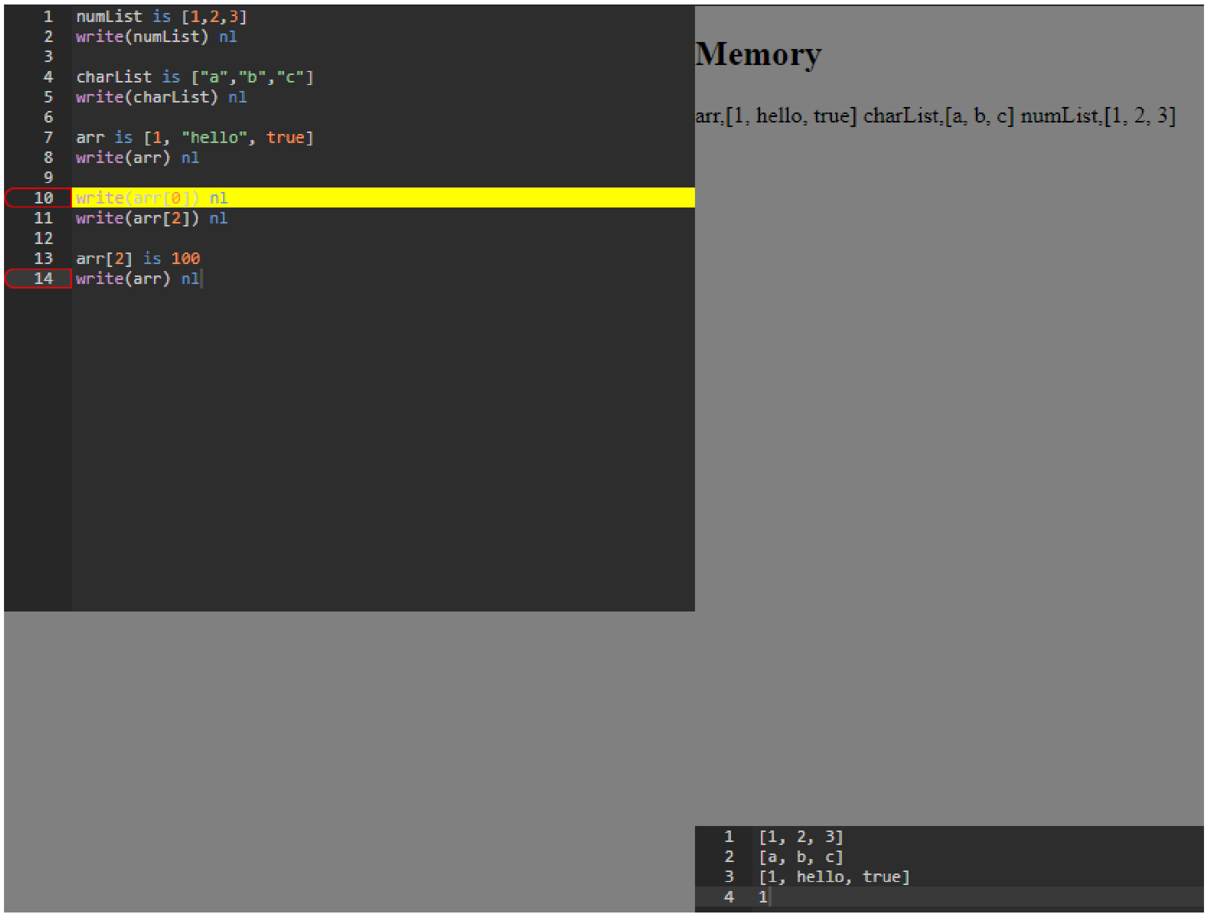


```
numList is [1,2,3]
write(numList) nl

charList is ["a","b","c"]
write(charList) nl

arr is [1, "hello", true]
write(arr) nl

write(arr[0]) nl
write(arr[2]) nl

arr[2] is 100
write(arr) nl
```

Memory

arr,[1, hello, true] charList,[a, b, c] numList,[1, 2, 3]

```
1  [1, 2, 3]
2  [a, b, c]
```

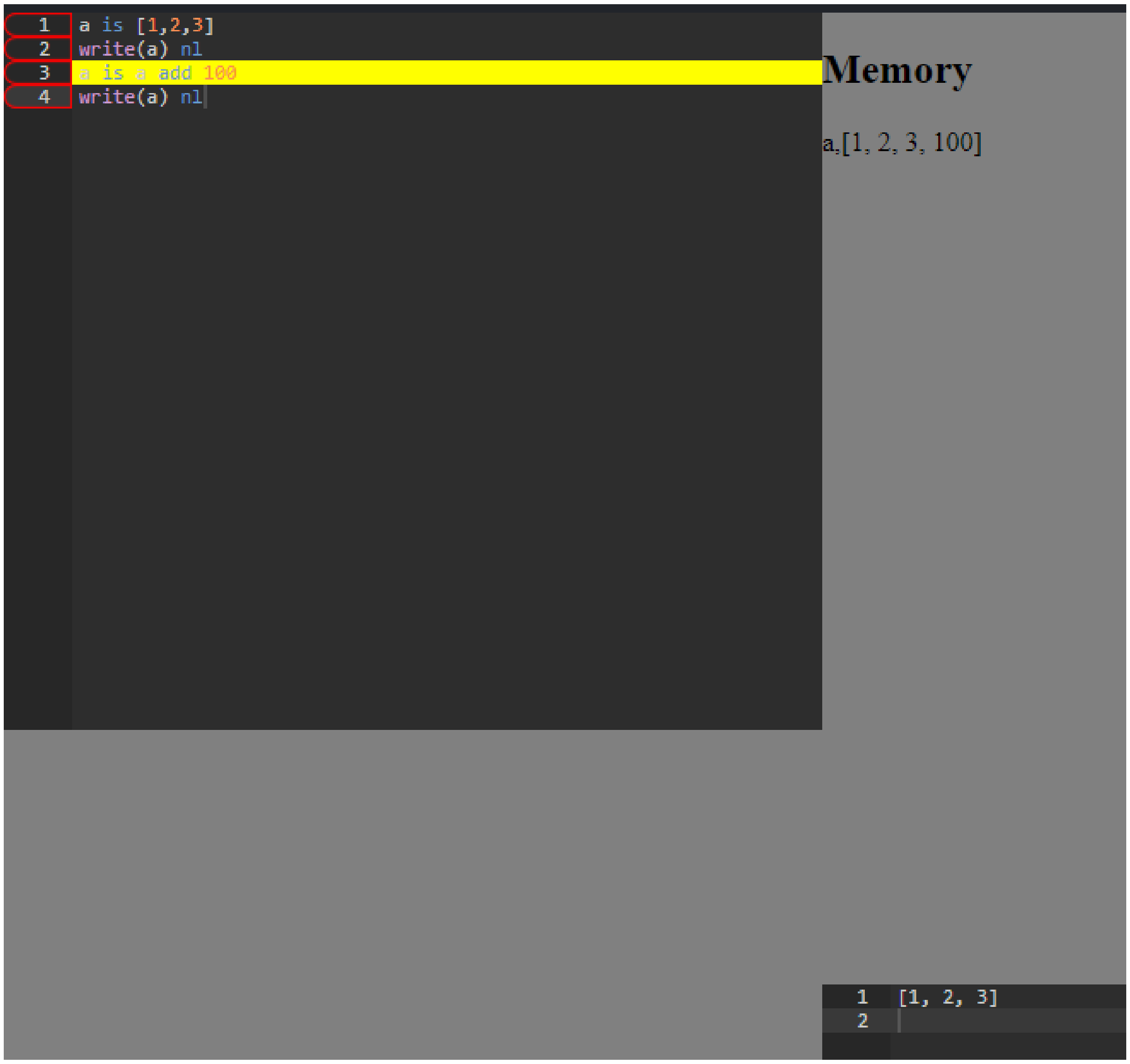


```
numList is [1,2,3]
write(numList) nl

charList is ["a","b","c"]
write(charList) nl

arr is [1, "hello", true]
write(arr) nl

write(arr[0]) nl
write(arr[2]) nl

arr[2] is 100
write(arr) nl
```

Memory

arr,[1, hello, true] charList,[a, b, c] numList,[1, 2, 3]

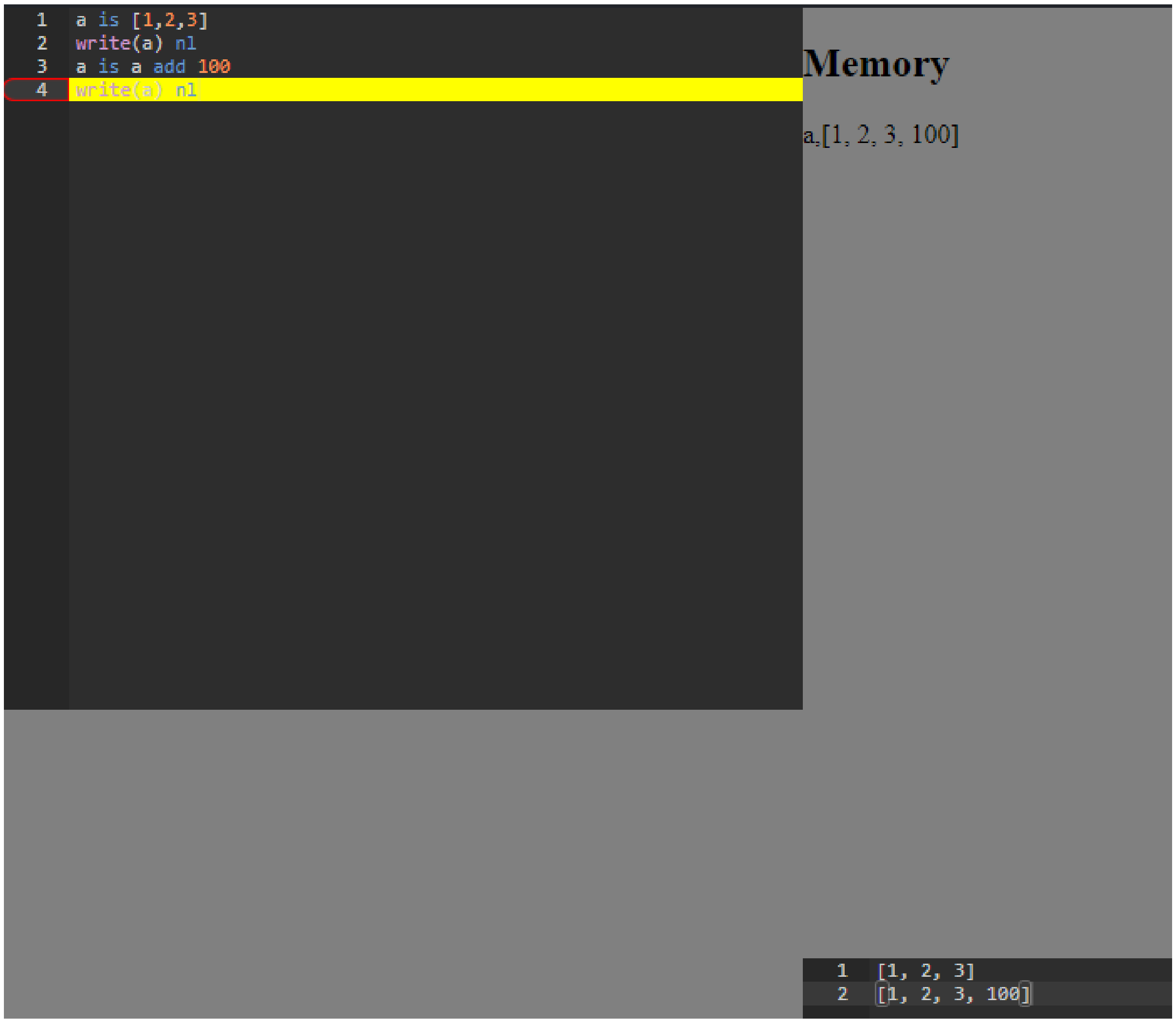
```
[1, 2, 3]
[a, b, c]
[1, hello, true]
1
```



```
a is [1,2,3]
write(a) nl
a is a add 100
write(a) nl
```

a [1, 2, 3, 100]

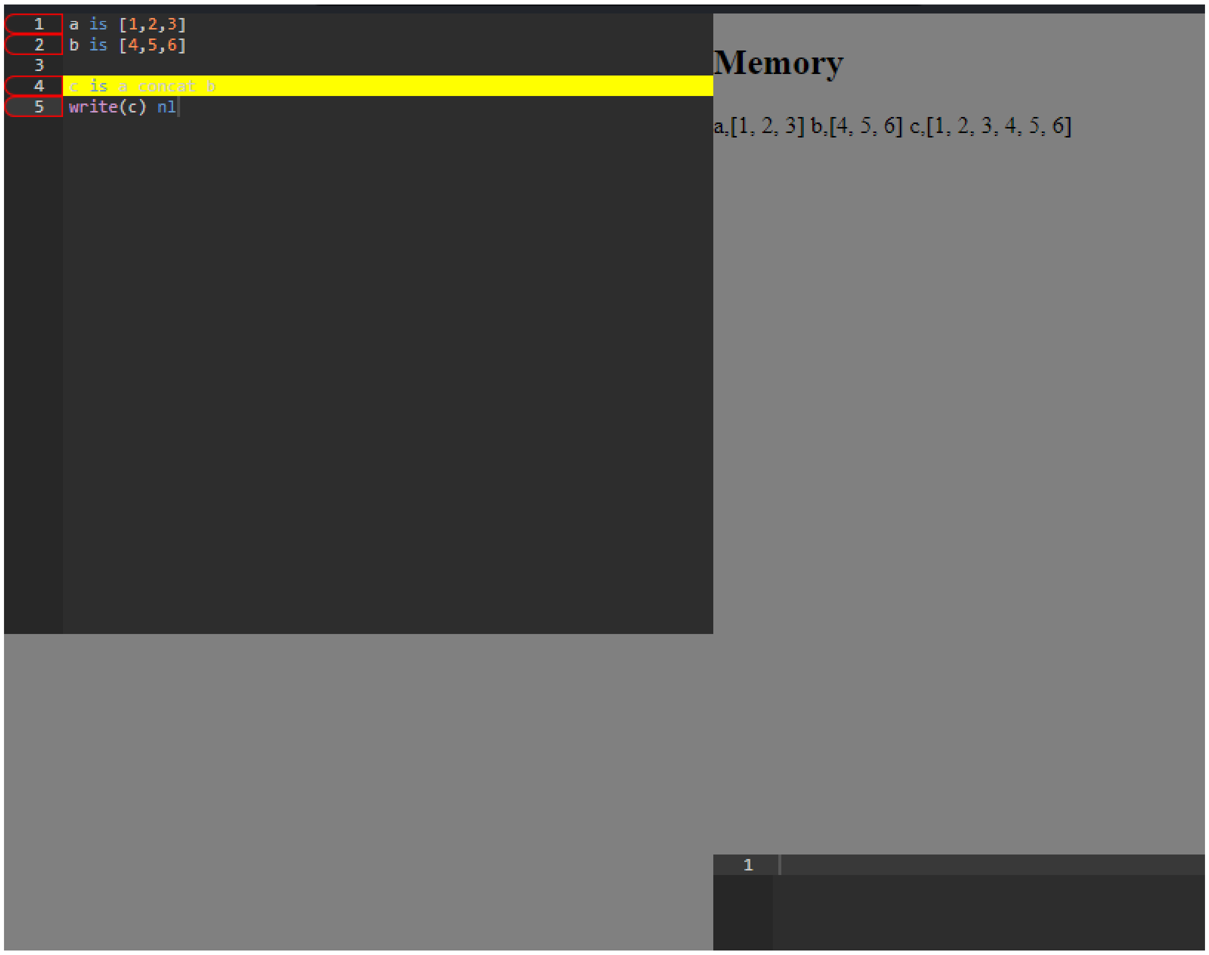
```
[1, 2, 3]
```



```
a is [1,2,3]
write(a) nl
a is a add 100
write(a) nl
```

```
a,[1, 2, 3, 100]
```

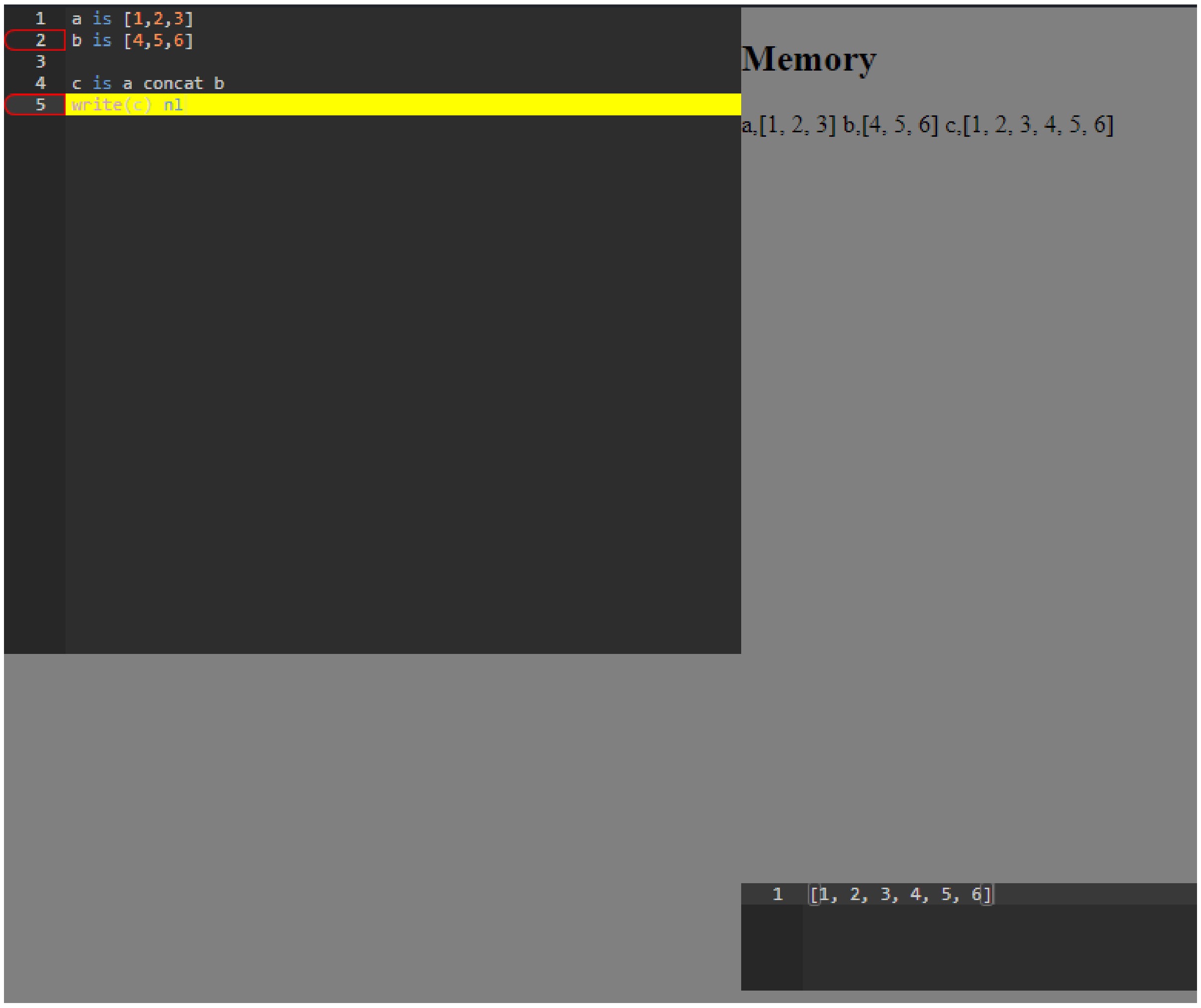
```
[1, 2, 3]
[1, 2, 3, 100]
```



```
a is [1,2,3]
b is [4,5,6]

c is a concat b
write(c) nl
```

Memory

a,[1, 2, 3] b,[4, 5, 6] c,[1, 2, 3, 4, 5, 6]

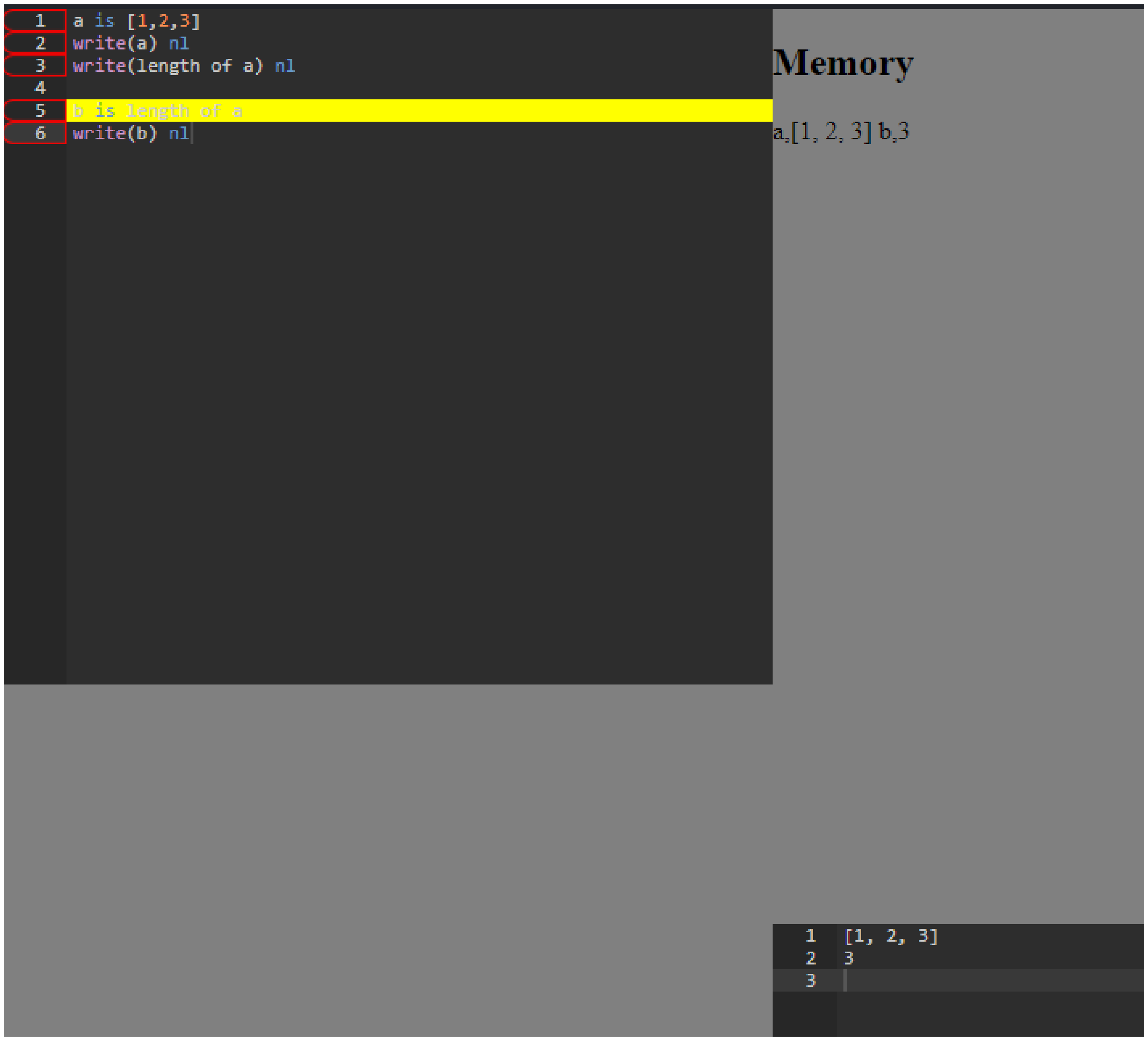


```
a is [1,2,3]
b is [4,5,6]

c is a concat b
write(c) nl
```

Memory

a,[1, 2, 3] b,[4, 5, 6] c,[1, 2, 3, 4, 5, 6]

```
[1, 2, 3, 4, 5, 6]
```

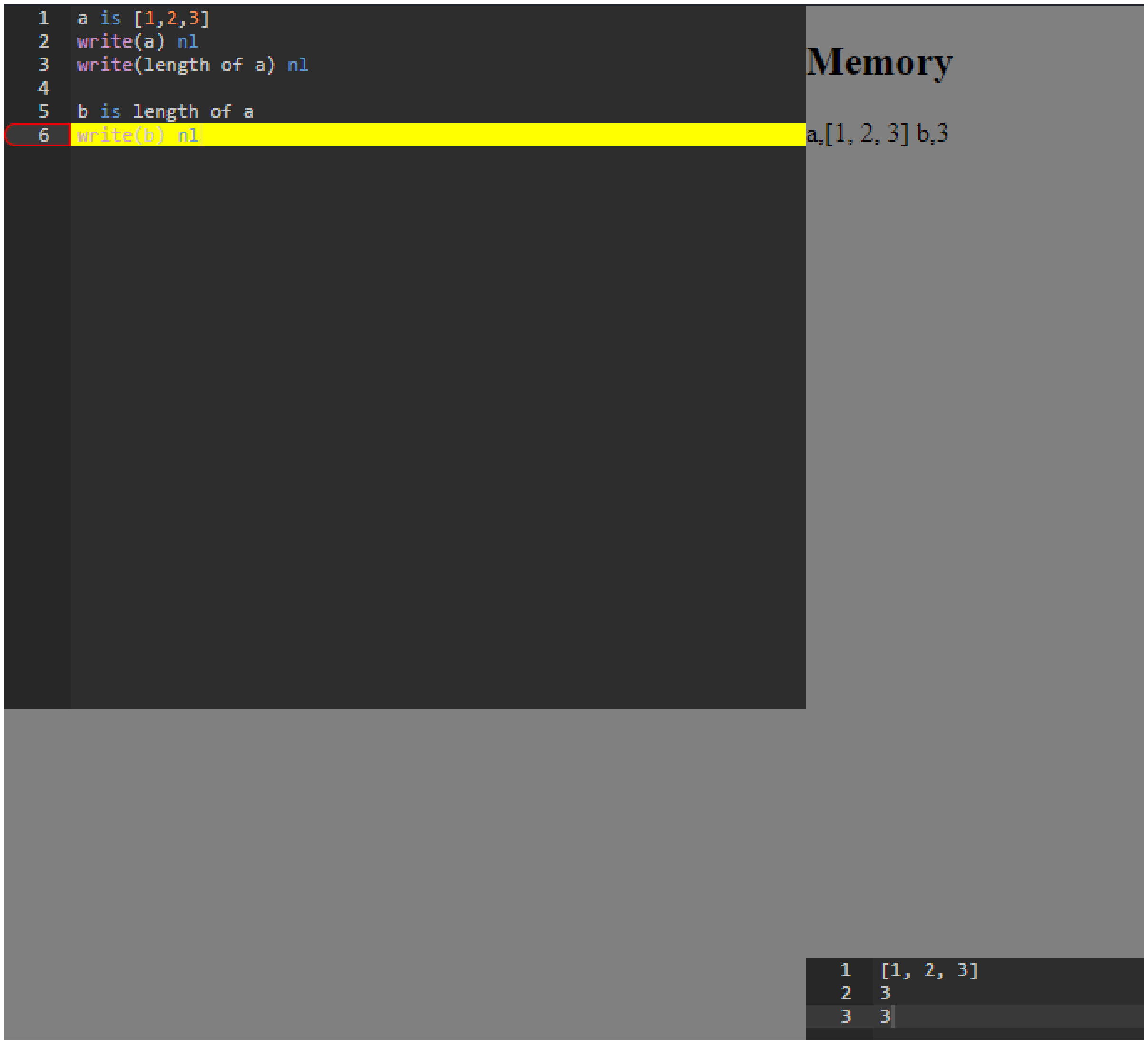


```
a is [1,2,3]
write(a) nl
write(length of a) nl

b is length of a
write(b) nl
```

Memory

a,[1, 2, 3] b,3

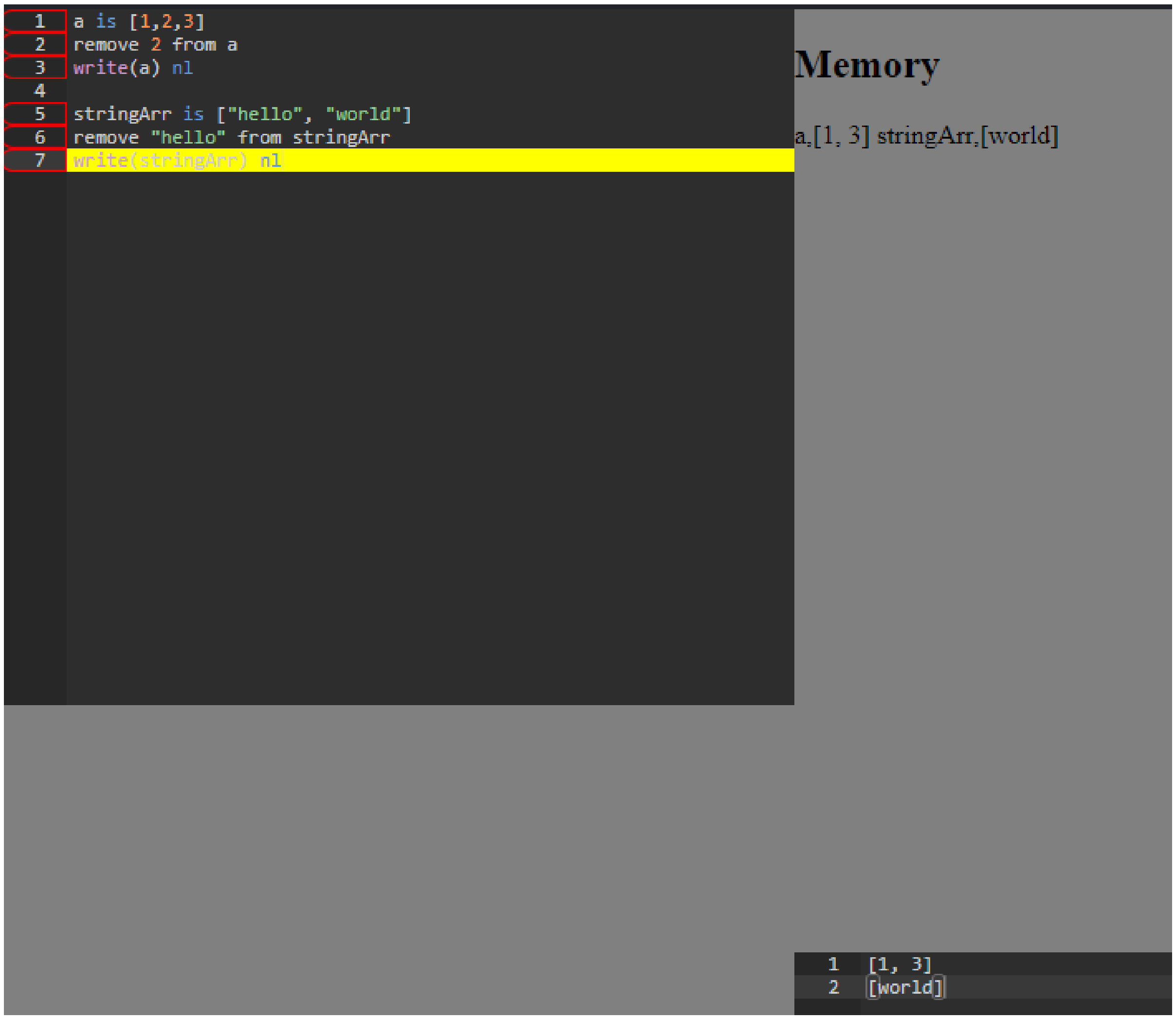
```
[1, 2, 3]
3
```



```
a is [1,2,3]
write(a) nl
write(length of a) nl

b is length of a
write(b) nl
```

a,[1, 2, 3] b,3

```
[1, 2, 3]
3
3
```

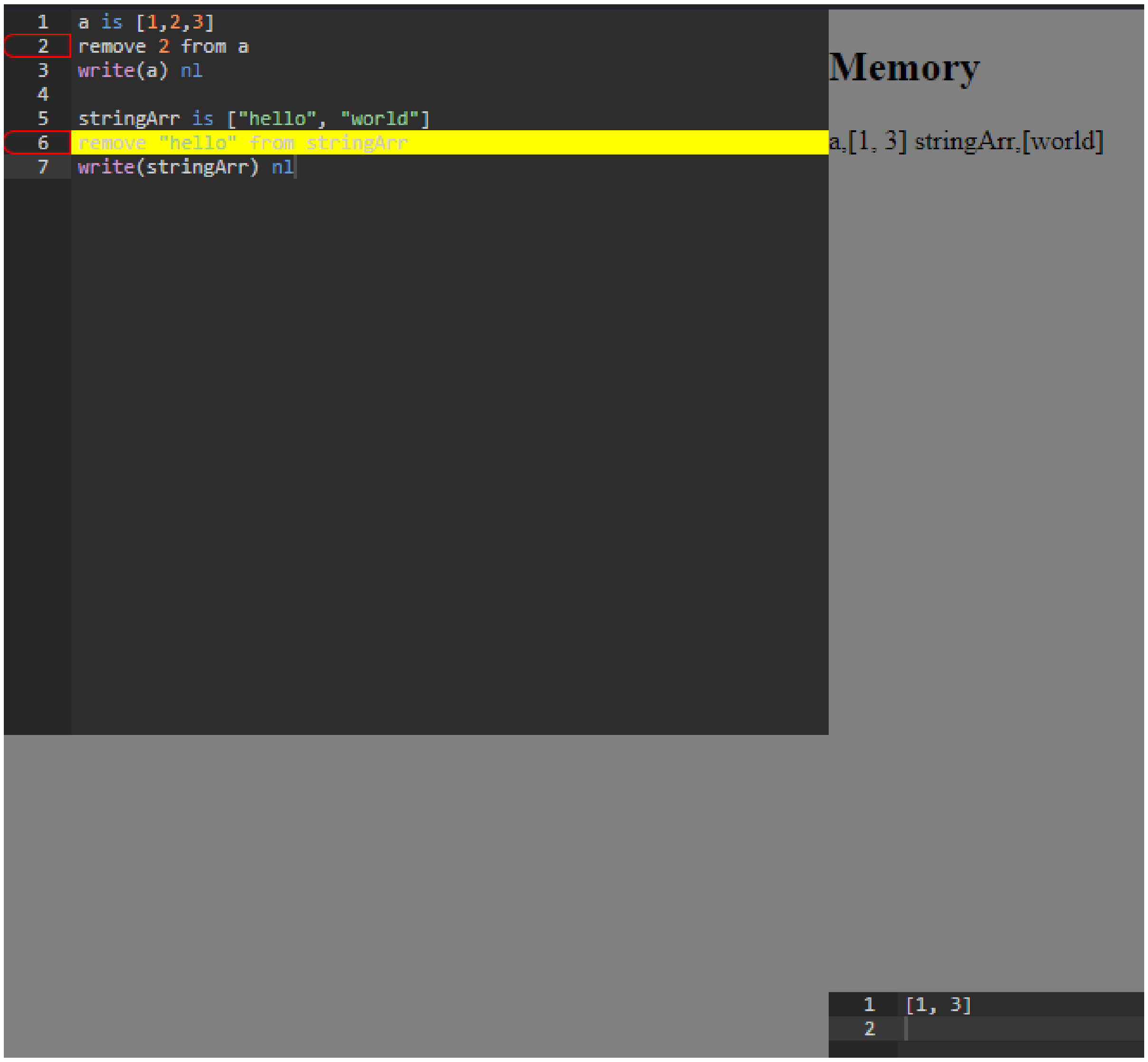


```
a is [1,2,3]
remove 2 from a
write(a) nl

stringArr is ["hello", "world"]
remove "hello" from stringArr
write(stringArr) nl
```

Memory

a,[1, 3] stringArr,[world]

```
[1, 3]
[world]
```



```
a is [1,2,3]
remove 2 from a
write(a) nl

stringArr is ["hello", "world"]
remove "hello" from stringArr
write(stringArr) nl
```

```
a,[1, 3] stringArr,[world]
```

```
[1, 3]
```

Overall Result

Pass

Test Name : If Statements & Boolean Operators

Test Description

If statements should be able to correctly follow the right path based on the comparison of each if, otherwise if and otherwise statement, and such we should see the correct messages.

Within these comparisons the Boolean Operators should be working properly meaning AND and OR should work as expected, with either both or one of the statements needing to be true, respectively.

Ensure the breakpoints work as expected.

Test Data

`if-statement.st`

Expected Result

We should see the messages:

```
15 is a multiple of 3 and 5
n is 15 and m is 11
a is greater than 1
```

We should also stop at the conditions of the if statement.

Execution Steps

Code ran in testing version of START Web Application.

Step button used to advance the program past it's breakpoints.

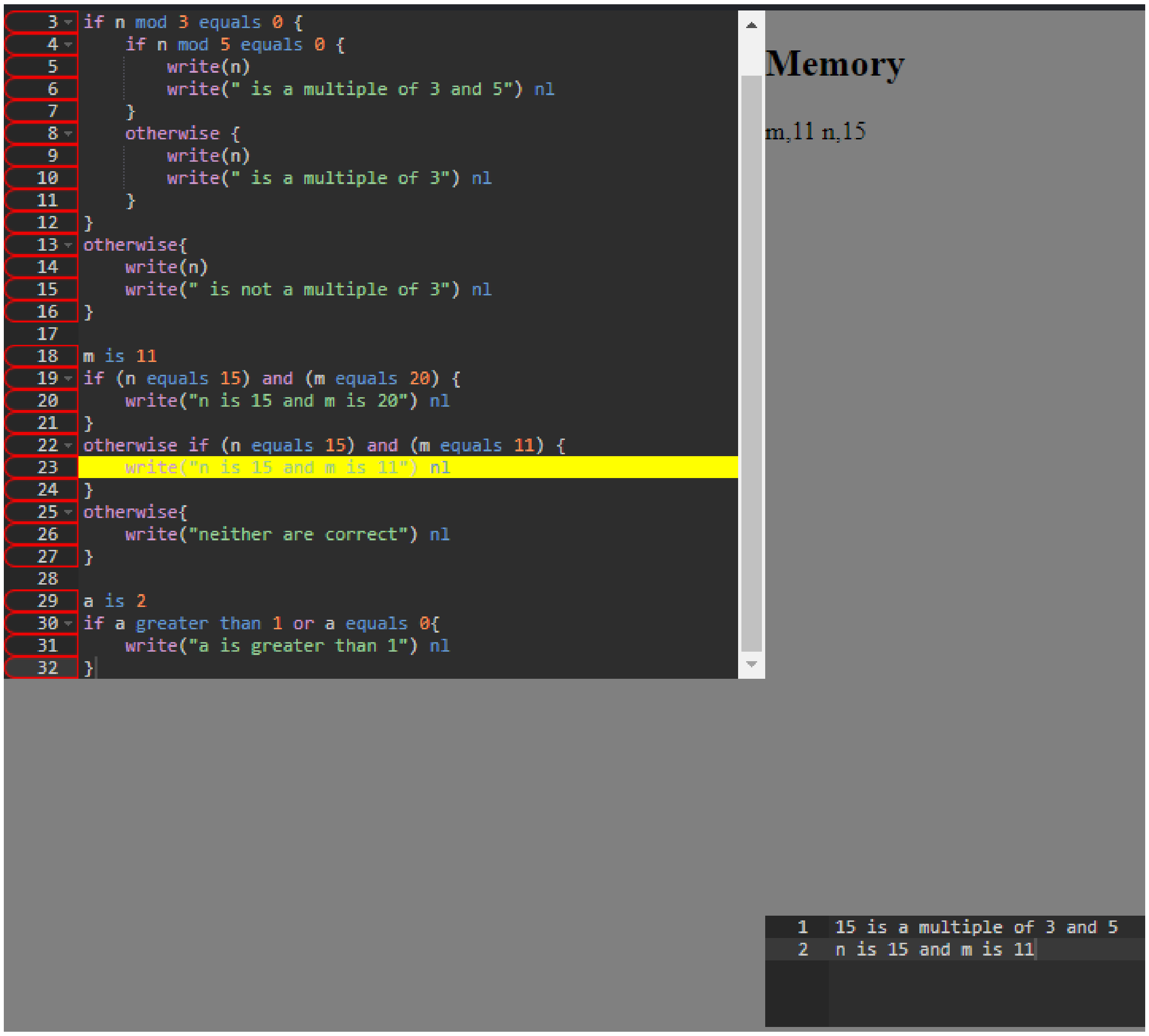
Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
    - All messages were seen.
    - There was 16 stopping points in the program.
    - Memory updated dynamically.
- Dynamic Mode:
    - All messages were seen.
    - The program stopped the correct number of times.

- Memory updated dynamically.

Evidence

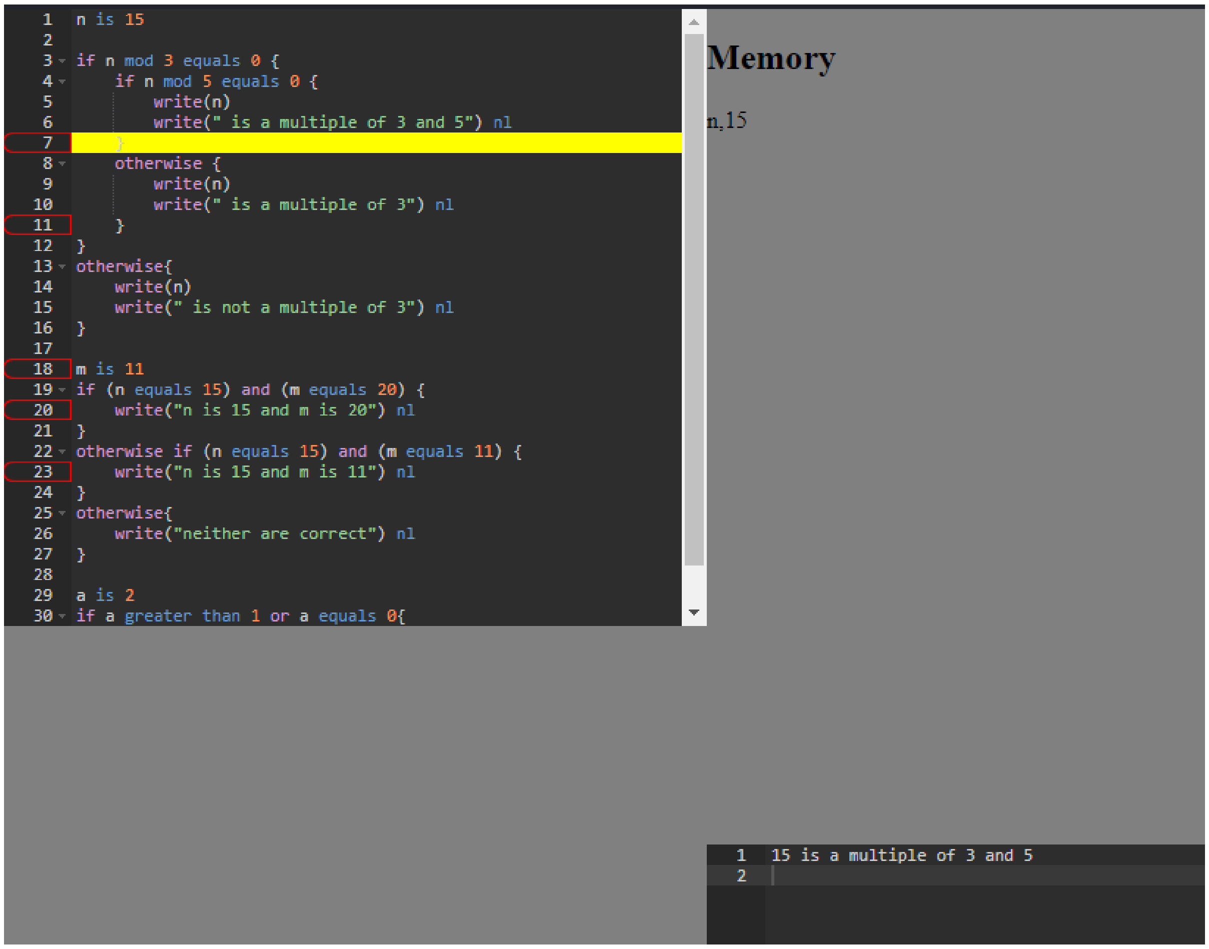


```
if n mod 3 equals 0 {
    if n mod 5 equals 0 {
        write(n)
        write(" is a multiple of 3 and 5") nl
    }
    otherwise {
        write(n)
        write(" is a multiple of 3") nl
    }
}
otherwise{
    write(n)
    write(" is not a multiple of 3") nl
}

m is 11
if (n equals 15) and (m equals 20) {
    write("n is 15 and m is 20") nl
}
otherwise if (n equals 15) and (m equals 11) {
    write("n is 15 and m is 11") nl
}
otherwise{
    write("neither are correct") nl
}

a is 2
if a greater than 1 or a equals 0{
    write("a is greater than 1") nl
}
```

Memory

m,11 n,15

```
15 is a multiple of 3 and 5
n is 15 and m is 11
```



```
n is 15

if n mod 3 equals 0 {
    if n mod 5 equals 0 {
        write(n)
        write(" is a multiple of 3 and 5") nl
    }
    otherwise {
        write(n)
        write(" is a multiple of 3") nl
    }
}
otherwise{
    write(n)
    write(" is not a multiple of 3") nl
}

m is 11
if (n equals 15) and (m equals 20) {
    write("n is 15 and m is 20") nl
}
otherwise if (n equals 15) and (m equals 11) {
    write("n is 15 and m is 11") nl
}
otherwise{
    write("neither are correct") nl
}

a is 2
if a greater than 1 or a equals 0{
```

Memory

n,15

```
15 is a multiple of 3 and 5
```

Overall Result

Pass

Test Name : While Loops

Test Description

The loop should be able to run given a certain provided condition is true, and stop once the condition is broken.

Ensure the breakpoints work as expected.

Test Data

`while-loop.st`

Expected Result

We should see the current state of `n` while it is less than 3, and each time we should see the state of m, as well as seeing `m` reset for each iteration.

There should be (68 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

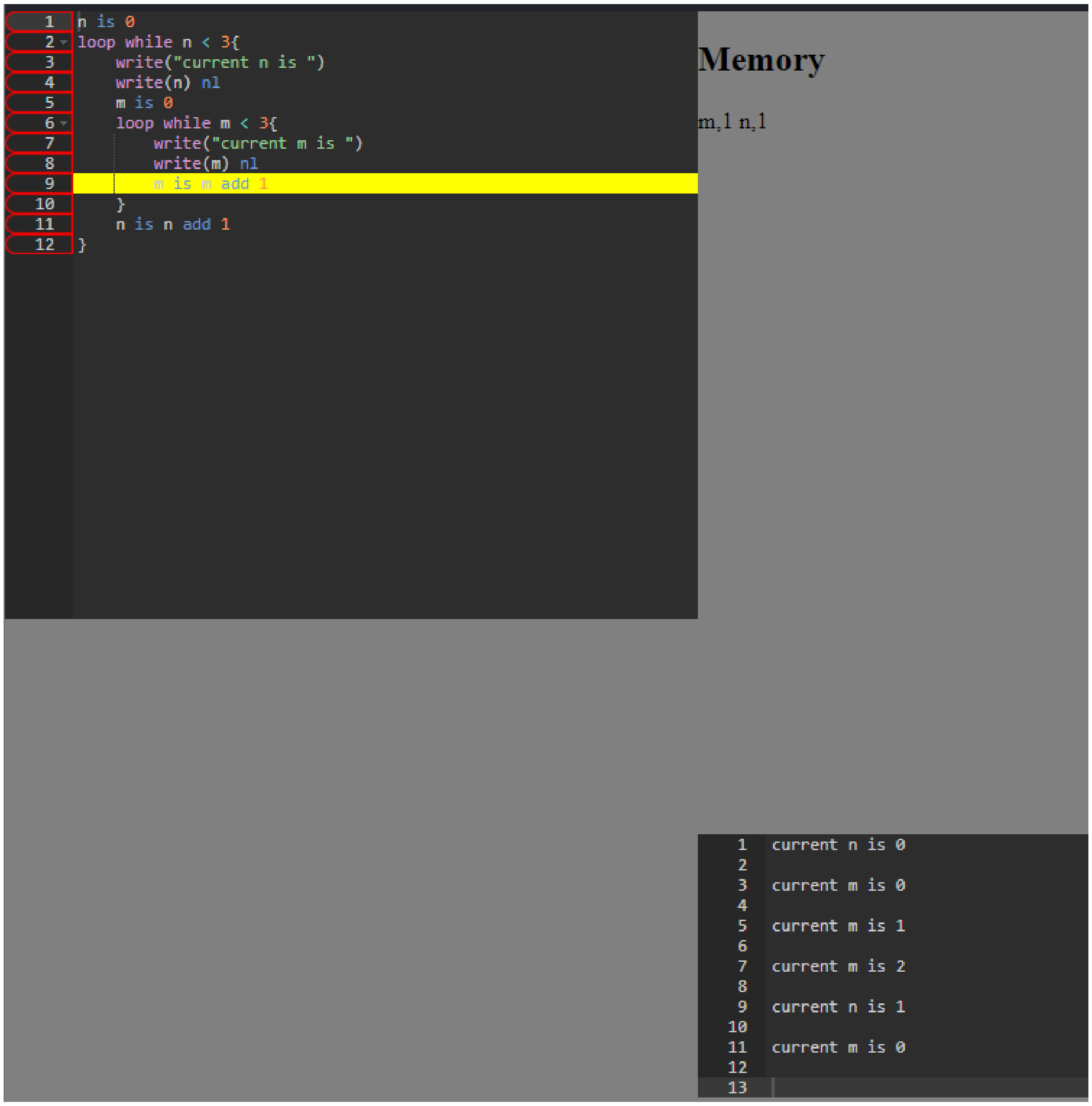
Step button used to advance the program past it’s breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
  - All variables were of correct value when seen.
  - There was 68 stopping points in the program.
  - Memory updated dynamically.
- Dynamic Mode:
  - All variables were of correct value when seen.
  - The program stopped the correct number of times.
  - Memory updated dynamically.

Evidence

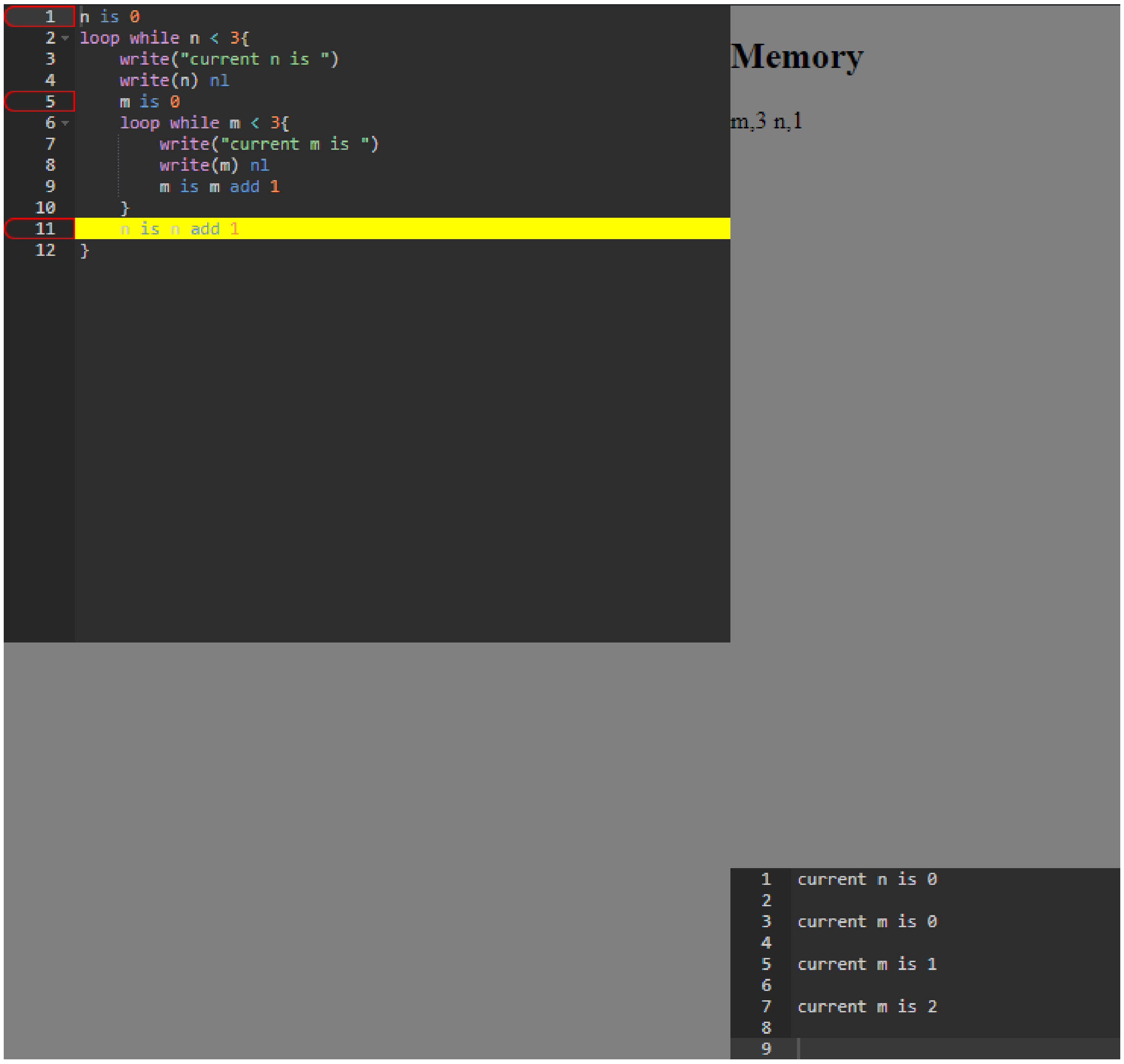


```
n is 0
loop while n < 3{
    write("current n is ")
    write(n) nl
    m is 0
    loop while m < 3{
        write("current m is ")
        write(m) nl
        m is m add 1
    }
    n is n add 1
}
```

Memory

m,1 n,1

```
current n is 0

current m is 0

current m is 1

current m is 2

current n is 1

current m is 0

```



```
n is 0
loop while n < 3{
    write("current n is ")
    write(n) nl
    m is 0
    loop while m < 3{
        write("current m is ")
        write(m) nl
        m is m add 1
    }
    n is n add 1
}
```

```
current n is 0

current m is 0

current m is 1

current m is 2

```

Overall Result

Pass

Test Name : For Loops

Test Description

The loop should be able to move through each element of the array, as well as the sub elements of a matrix in this given case.

Ensure the breakpoints work as expected.

Test Data

`for-loop.st`

Expected Result

We expect to see the numbers 1 to 6 sequentially as the embedded loops work to extract the elements of the matrix.

There should be (26 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

Step button used to advance the program past it's breakpoints.

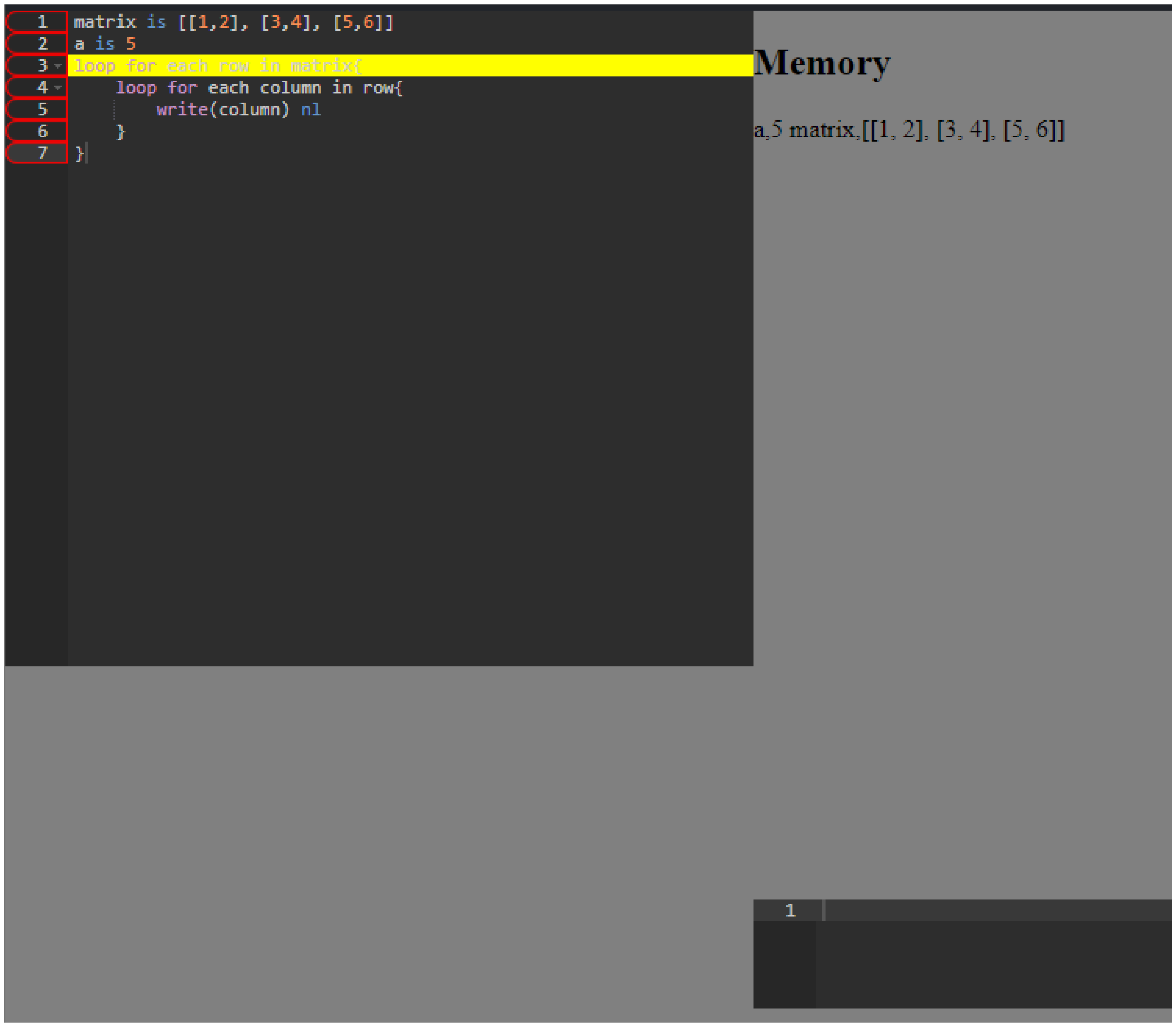
Examine output to ensure variables are of correct value at the end of the program.

Actual Result

We see the numbers of the matrix printed out in order correctly.

- Line-By-Line Mode:
    - Output was correct.
    - There was more than 26 stopping points
    - Memory updated dynamically.
- Dynamic Mode:
    - Output was correct.
    - The program did not stop the correct number of times.
    - Memory updated dynamically.

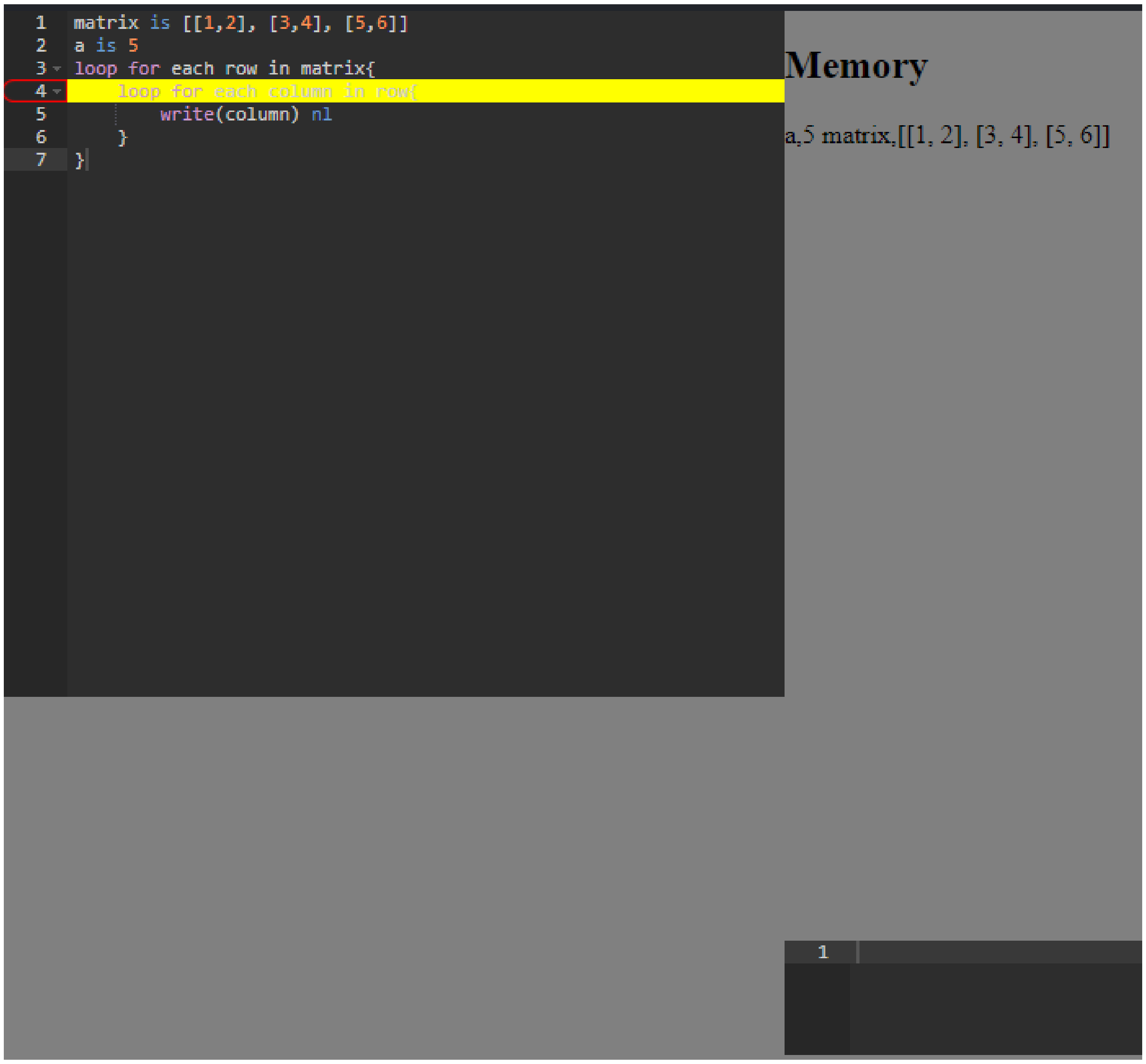
Evidence



```
matrix is [[1,2], [3,4], [5,6]]
a is 5
loop for each row in matrix{
    loop for each column in row{
        write(column) nl
    }
}
```

Memory

```
a,5 matrix,[[1, 2], [3, 4], [5, 6]]
```


```
matrix is [[1,2], [3,4], [5,6]]
a is 5
loop for each row in matrix{
    loop for each column in row{
        write(column) nl
    }
}
```

a,5 matrix,[[1, 2], [3, 4], [5, 6]]

Overall Result

Fail

Test Name : Functions

Test Description

We should be able to define a function, as well as return from one, call a function within a function, and print from a function.

Ensure the breakpoints work as expected.

Test Data

`functions.st`

Expected Result

We expect to see `Hello World` printed when `main` is called correctly, and then see 7 when the `addup` function is called within main and the values passed are returned added together.

There should be (9 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

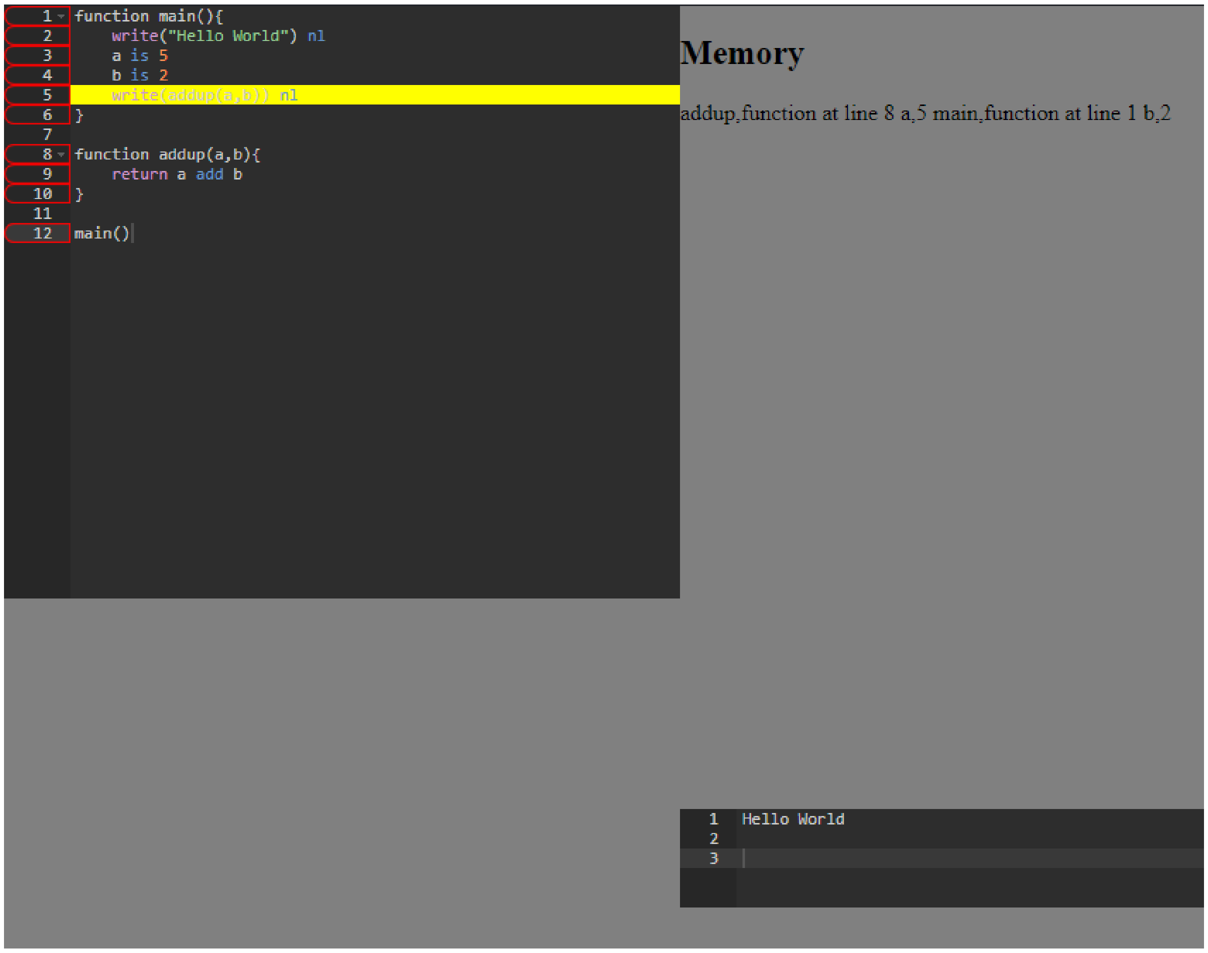
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

The code stopped out of order, stopping on line 5 instead of entering the function.

Evidence

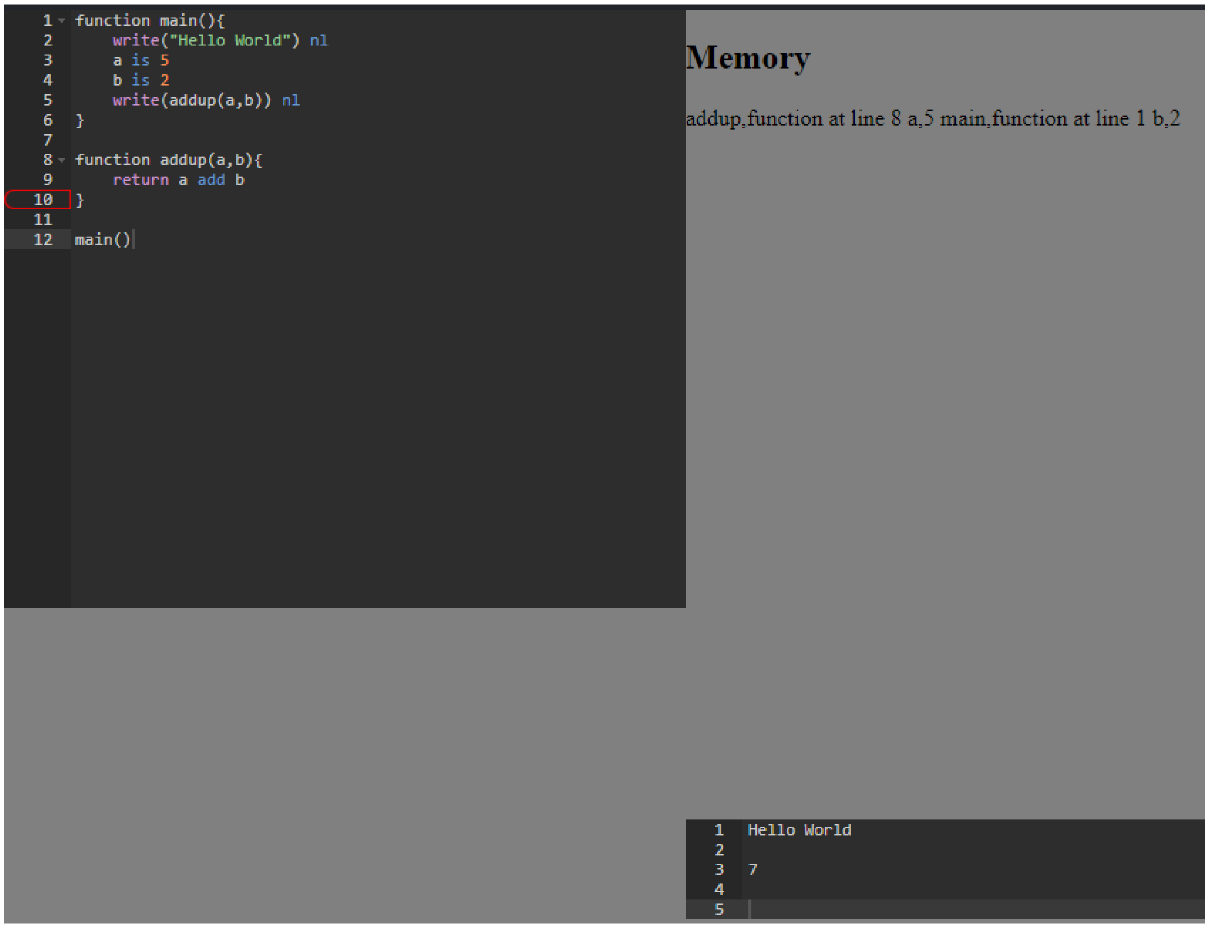


```
function main(){
    write("Hello World") nl
    a is 5
    b is 2
    write(addup(a,b)) nl
}

function addup(a,b){
    return a add b
}

main()
```

Memory

addup,function at line 8 a,5 main,function at line 1 b,2

```
Hello World
```



```
function main(){
    write("Hello World") nl
    a is 5
    b is 2
    write(addup(a,b)) nl
}

function addup(a,b){
    return a add b
}

main()
```

Memory

addup,function at line 8 a,5 main,function at line 1 b,2

```
Hello World

7


```

Overall Result

Fail

Test Name : Remove All

Test Description

We should be able to remove all instances of a given number from a given array using a built in operation. No other indexes should be removed.

Ensure the breakpoints work as expected.

Test Data

`remove-all.st`

Expected Result

We expect to see the original array within any 1’s within the array outputted.

There should be (3 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

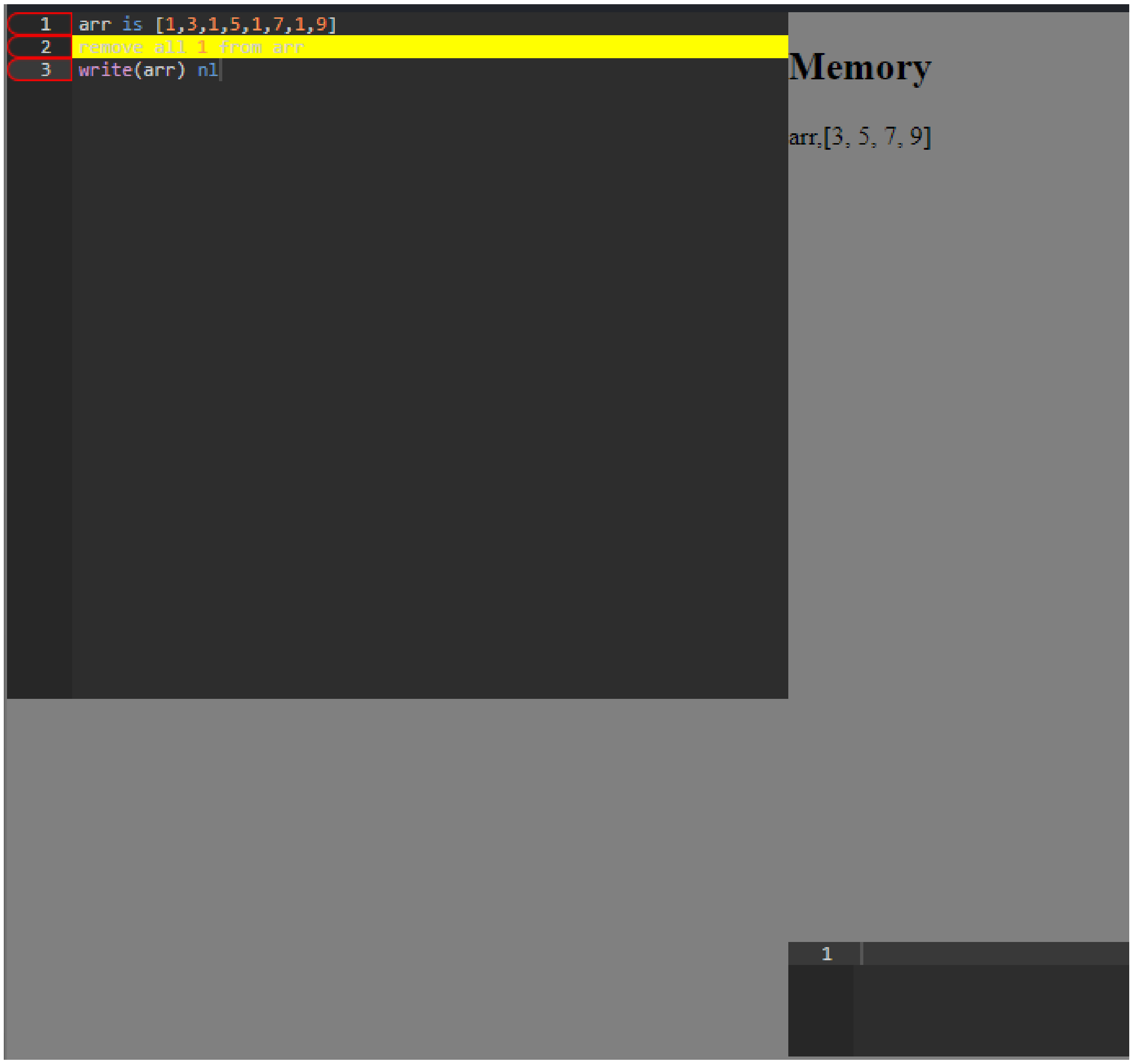
Step button used to advance the program past it’s breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
  - All variables were of correct value.
  - There was 3 stopping points.
  - Memory updated dynamically.
- Dynamic Mode:
  - All variables were of correct value.
  - The program stopped the correct number of times.
  - Memory updated dynamically.

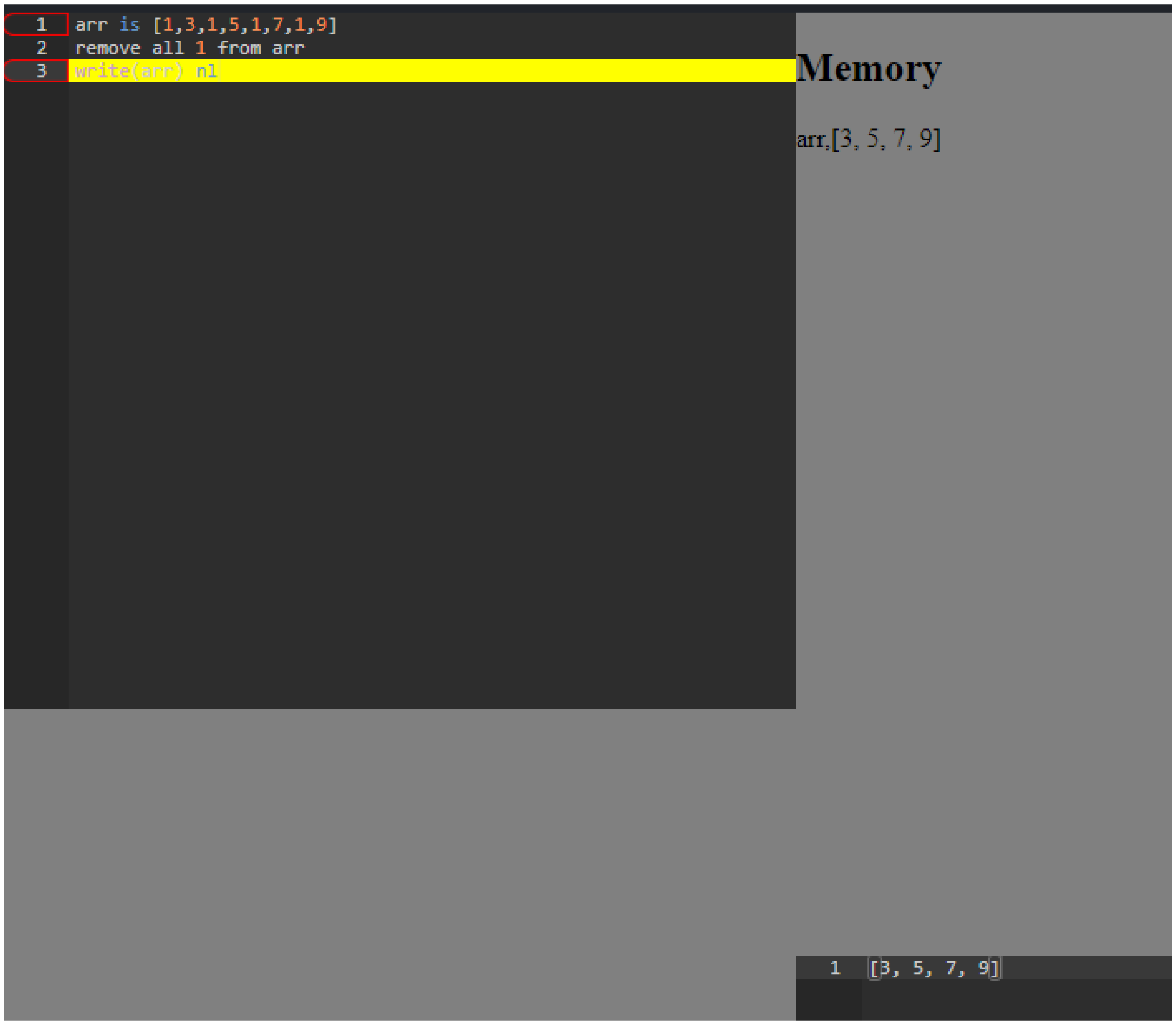
Evidence



```
arr is [1,3,1,5,1,7,1,9]
remove all 1 from arr
write(arr) nl
```

Memory

arr,[3, 5, 7, 9]



```
arr is [1,3,1,5,1,7,1,9]
remove all 1 from arr
write(arr) nl
```

Memory

arr,[3, 5, 7, 9]

```
[3, 5, 7, 9]
```

Overall Result

Pass

Test Name : String Indexing

Test Description

We expect that a character from a string can be stored in a variable by using its index.

Ensure the breakpoints work as expected.

Test Data

`string-dex.st`

Expected Result

We expect to see the correct index value printed out.

There should be (8 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

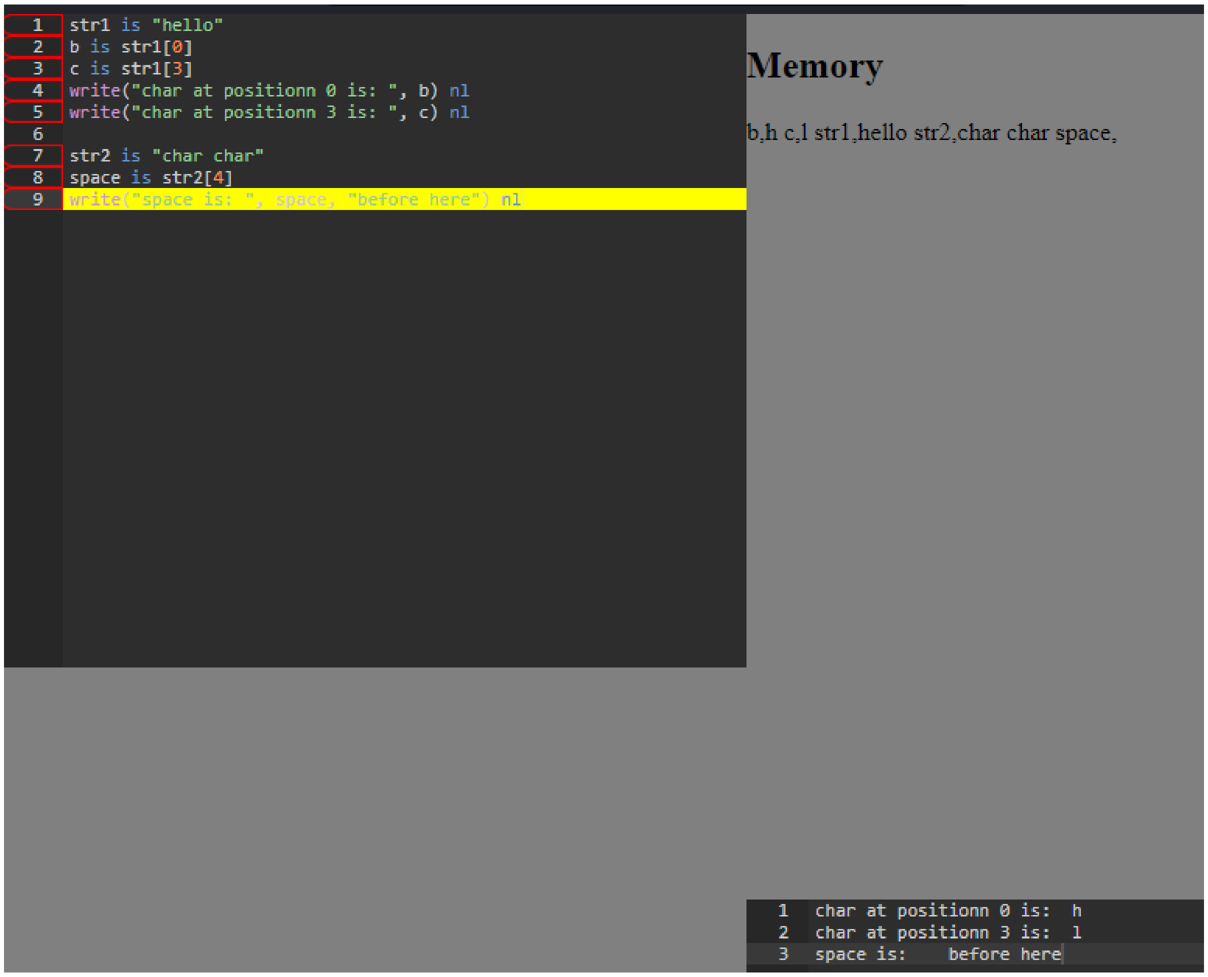
Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

Actual Result

- Line-By-Line Mode:
  - The correct index was seen.
  - There was 8 stopping points.
  - Memory updated dynamically.
- Dynamic Mode:
  - The correct index was seen.
  - The program stopped the correct number of times.
  - Memory updated dynamically.

Evidence

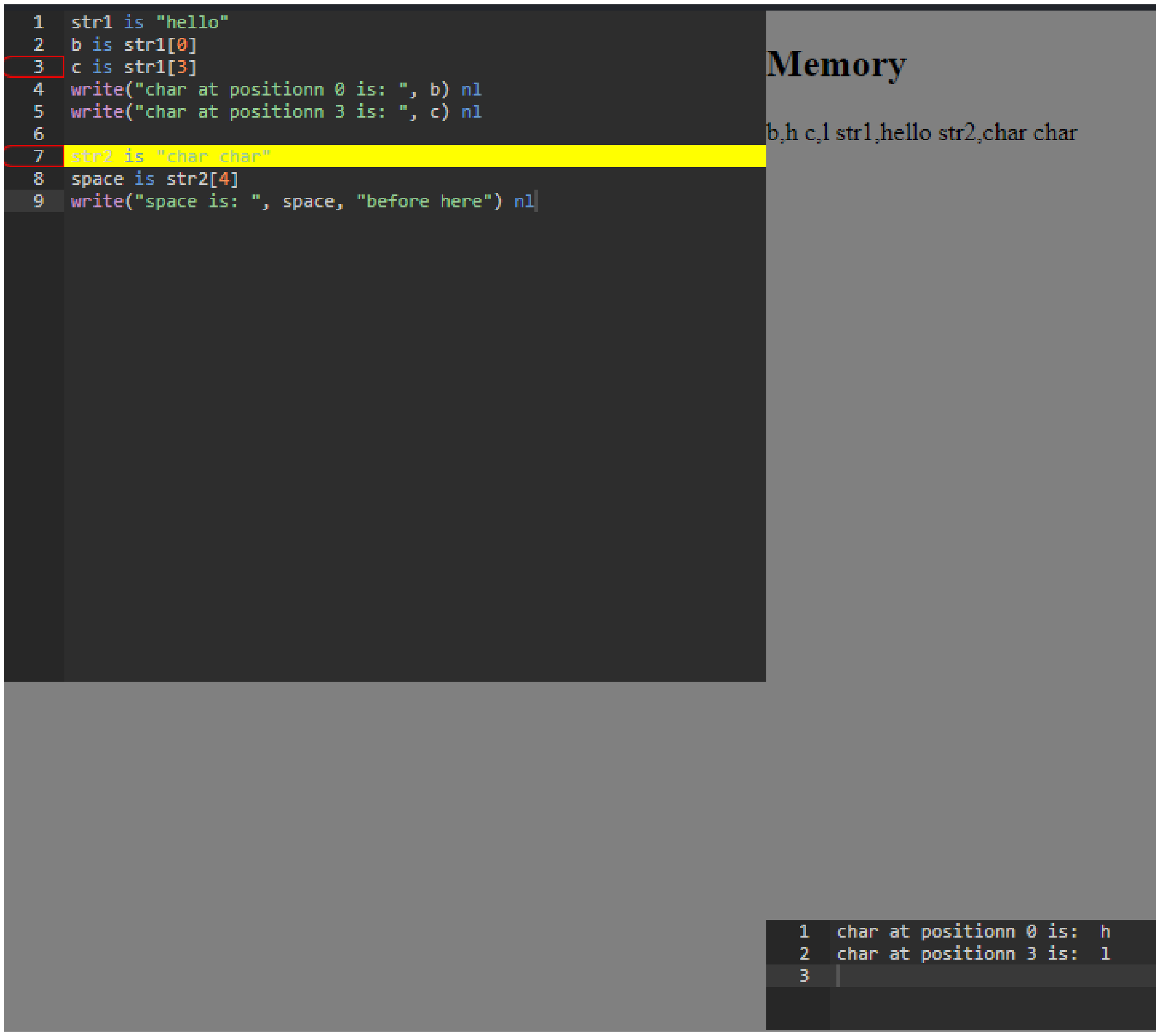


```
str1 is "hello"
b is str1[0]
c is str1[3]
write("char at positionn 0 is: ", b) nl
write("char at positionn 3 is: ", c) nl

str2 is "char char"
space is str2[4]
write("space is: ", space, "before here") nl
```

Memory

b,h c,l str1,hello str2,char char space,

```
1  char at positionn 0 is:  h
2  char at positionn 3 is:  l
3  space is:     before here
```



```
str1 is "hello"
b is str1[0]
c is str1[3]
write("char at positionn 0 is: ", b) nl
write("char at positionn 3 is: ", c) nl

str2 is "char char"
space is str2[4]
write("space is: ", space, "before here") nl
```

b,h c,l str1,hello str2,char char

```
char at positionn 0 is:  h
char at positionn 3 is:  l
```

Overall Result

Pass

Test Name : Comments

Test Description

Comments should be ignored completely by the program.

Ensure the breakpoints work as expected.

Test Data

`comments.st`

Expected Result

There should be (13 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

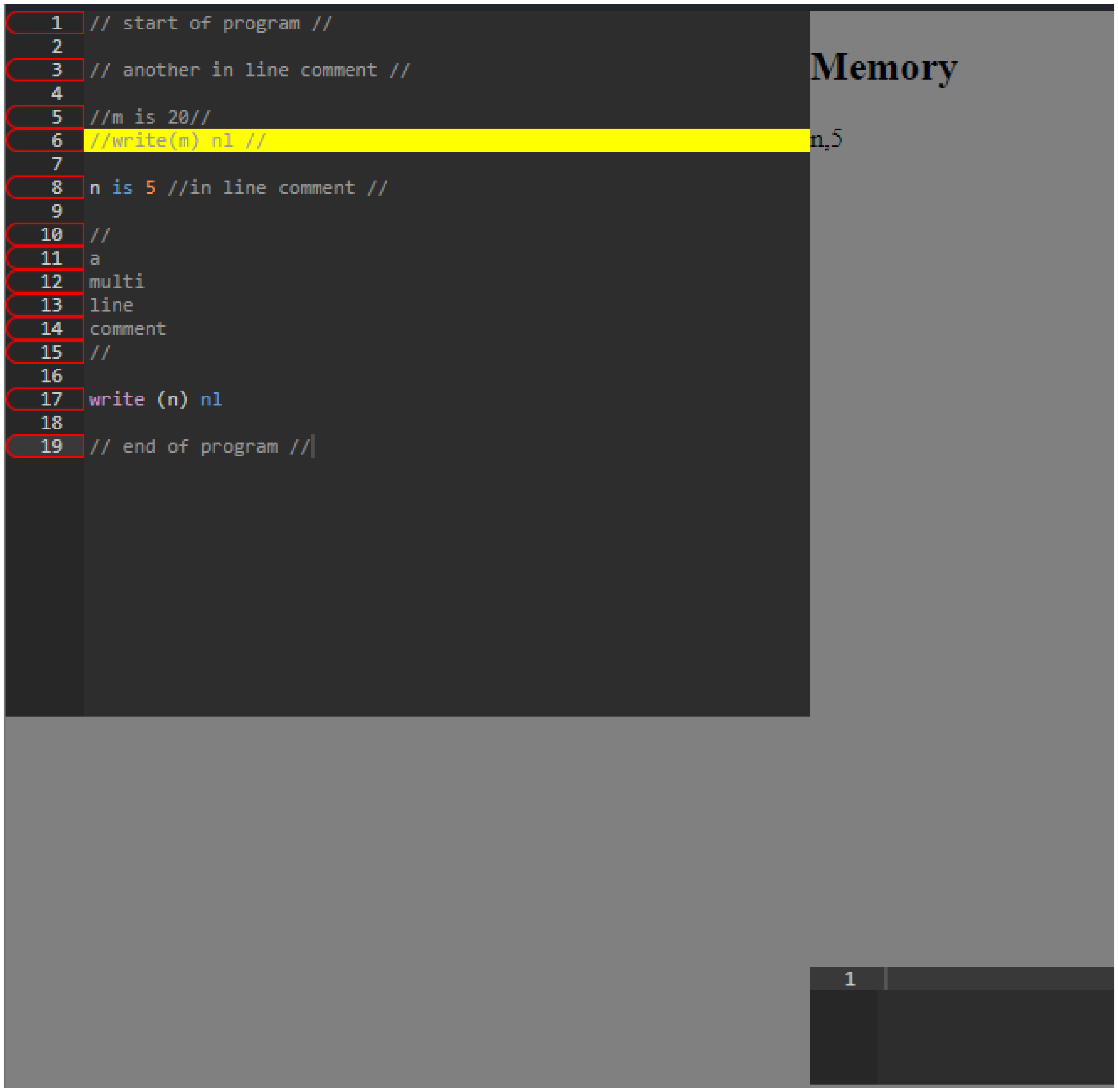
Step button used to advance the program past it’s breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

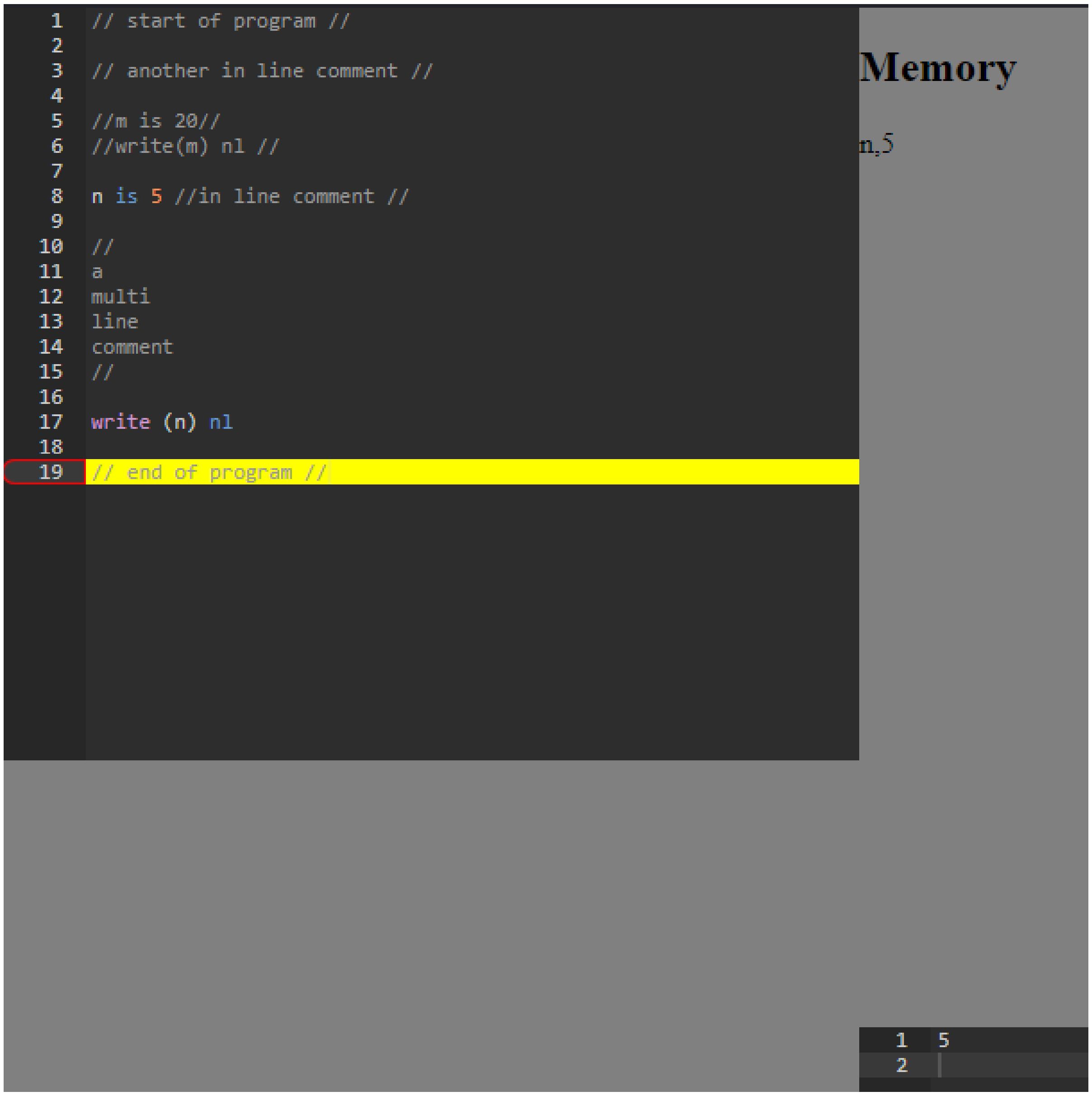
Actual Result

- Line-By-Line Mode:
  - There was 13 stopping points.
  - Memory updated dynamically.
- Dynamic Mode:
  - The program stopped the correct number of times.
  - Memory updated dynamically.

Evidence



```
// start of program //

// another in line comment //

//m is 20//
//write(m) nl //

n is 5 //in line comment //

//
a
multi
line
comment
//

write (n) nl

// end of program //
```



```
// start of program //

// another in line comment //

//m is 20//
//write(m) nl //

n is 5 //in line comment //

//
a
multi
line
comment
//

write (n) nl

// end of program //
```

```
5
```

Overall Result

Pass

Test Name : Error Generation Outputted

Test Description

An error should be shown when the code written contains errors.

Test Data

`error.st`

Expected Result

We get an assignment error as `i` is not defined.

Execution Steps

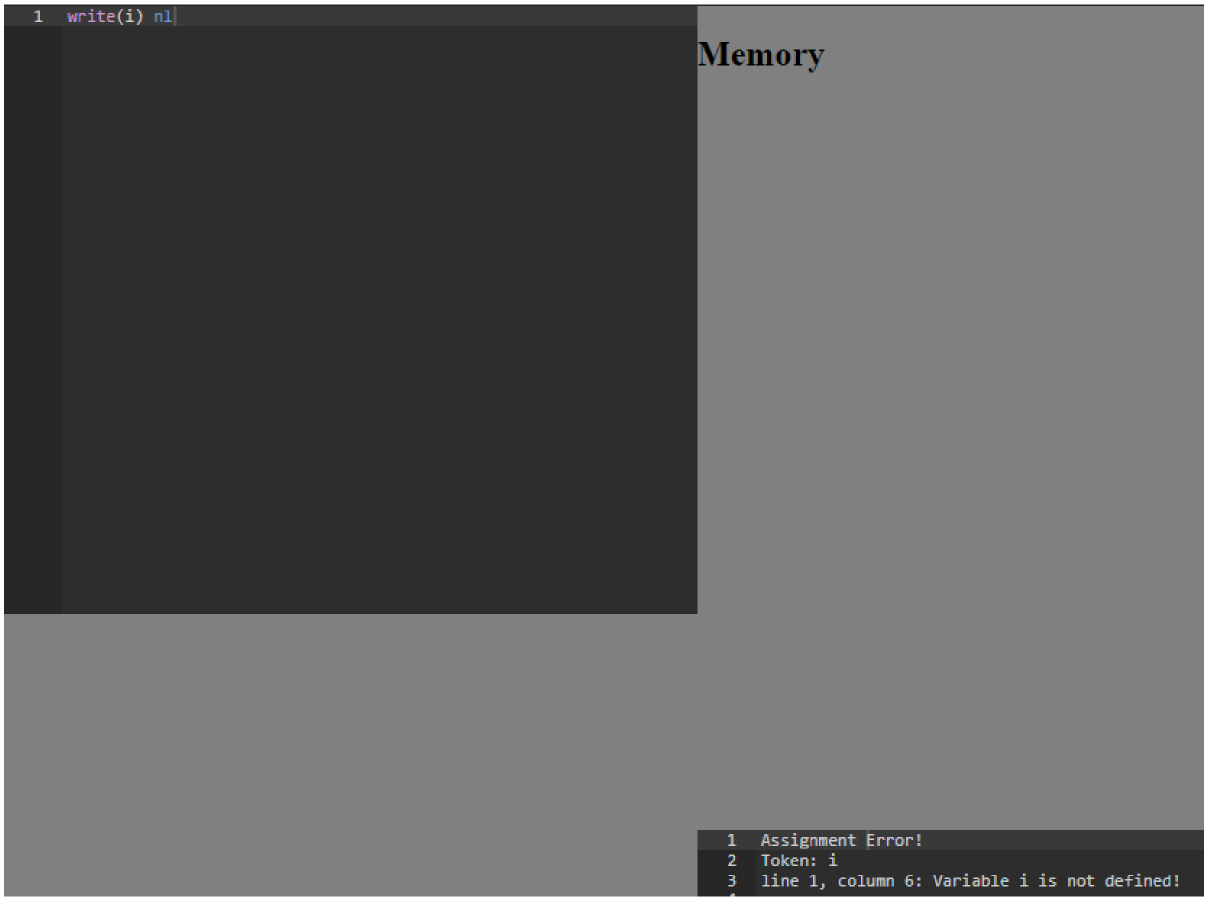
Code ran in testing version of START Web Application.

Examine output to ensure error is present.

Actual Result

```
Assignment Error!
Token: i
line 1, column 6: Variable i is not defined!
```

Evidence



```
write(i) nl
```

```
Assignment Error!
Token: i
line 1, column 6: Variable i is not defined!
```

Overall Result

Pass

Identified Defects

1. For loops are stopping twice on the `loop for` line when the line also contains the opening curly brace.
    - The following piece of code was used to fix this bug

```
int line = ctx.start.getLine();
boolean lineAdded = false;
if (breakPointArr.contains(ctx.start.getLine())){
    linesStoppedOnSoFar.add(line);
    lineAdded = true;
    breakpoint(line);
```

```
}

...

try {
    if (lineAdded){
        if (linesStoppedOnSoFar.contains(line)){
            linesStoppedOnSoFar.remove(linesStoppedOnSoFa
        }
    }
} catch (Exception e) {
    printLine(e.toString());
}
```

- We needed to just store our stop on the original for line, in case the opening brace was on the same line, we then remove it later to allow us to stop there again when looping
- Fixed by Adam Gray

2. Functions are visiting the lines of the program incorrectly, stopping on a line instead of entering a function, and then not visiting the last line of a function too.
   - The following piece of code was used to fix this bug

```
for (int i = startFunc; i < firstlineFunc; i++){
    if (breakPointArr.contains(i) && !linesStoppedOnSoFar
        int line = i;
        linesStoppedOnSoFar.add(line);
        breakpoint(line);
    }
}
```

   - The loop originally had `i = 0` which was leading to every line that had not been stopped on before the second function would end up being stopped on until it got to the function lines. Changing it to `i = startFunc`, where `startFunc` is the first program line the function appears on, fixed the issue.

- Fixed by Adam Gray

Regression Testing

Test Name : For Loops

Test Description

The loop should be able to move through each element of the array, as well as the sub elements of a matrix in this given case.

Ensure the breakpoints work as expected.

Test Data

`for-loop.st`

Expected Result

We expect to see the numbers 1 to 6 sequentially as the embedded loops work to extract the elements of the matrix.

There should be (26 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

Step button used to advance the program past it's breakpoints.

Examine output to ensure variables are of correct value at the end of the program.

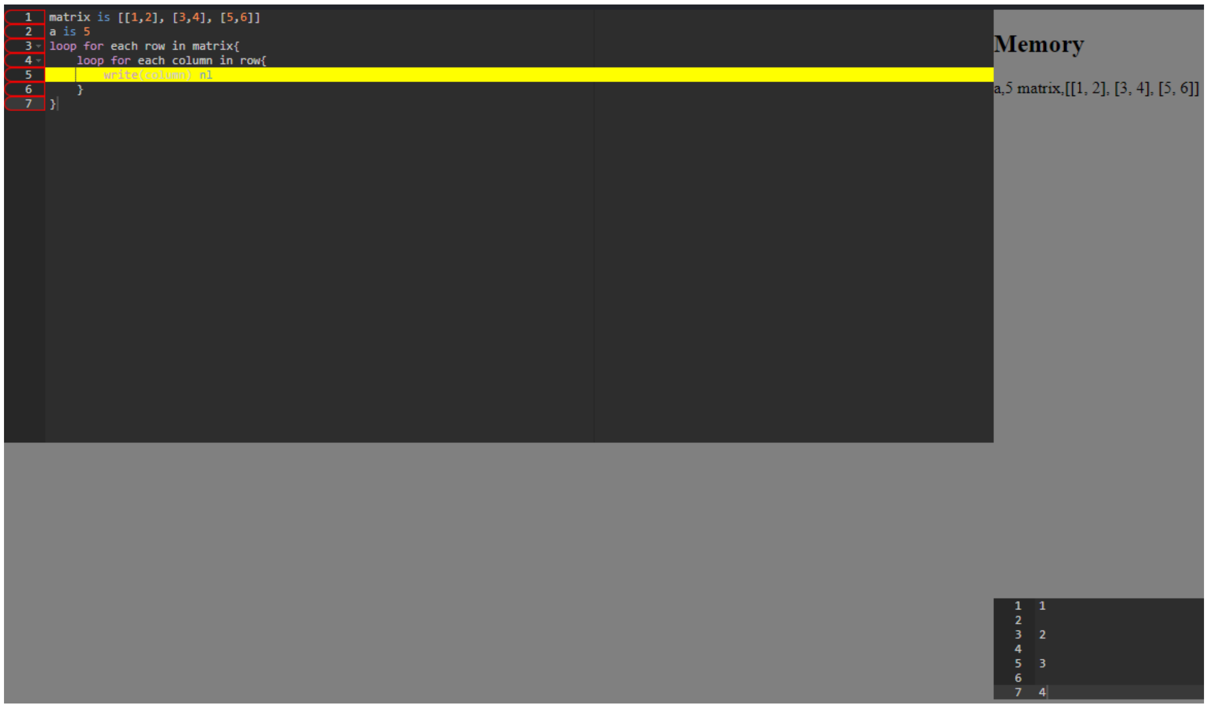
Actual Result

We see the numbers of the matrix printed out in order correctly.

- Line-By-Line Mode:
    - Output was correct.

    - There was 26 stopping points.
    - Memory updated dynamically.
- Dynamic Mode:
    - Output was correct.
    - The program stopped the correct number of times.
    - Memory updated dynamically.

Evidence

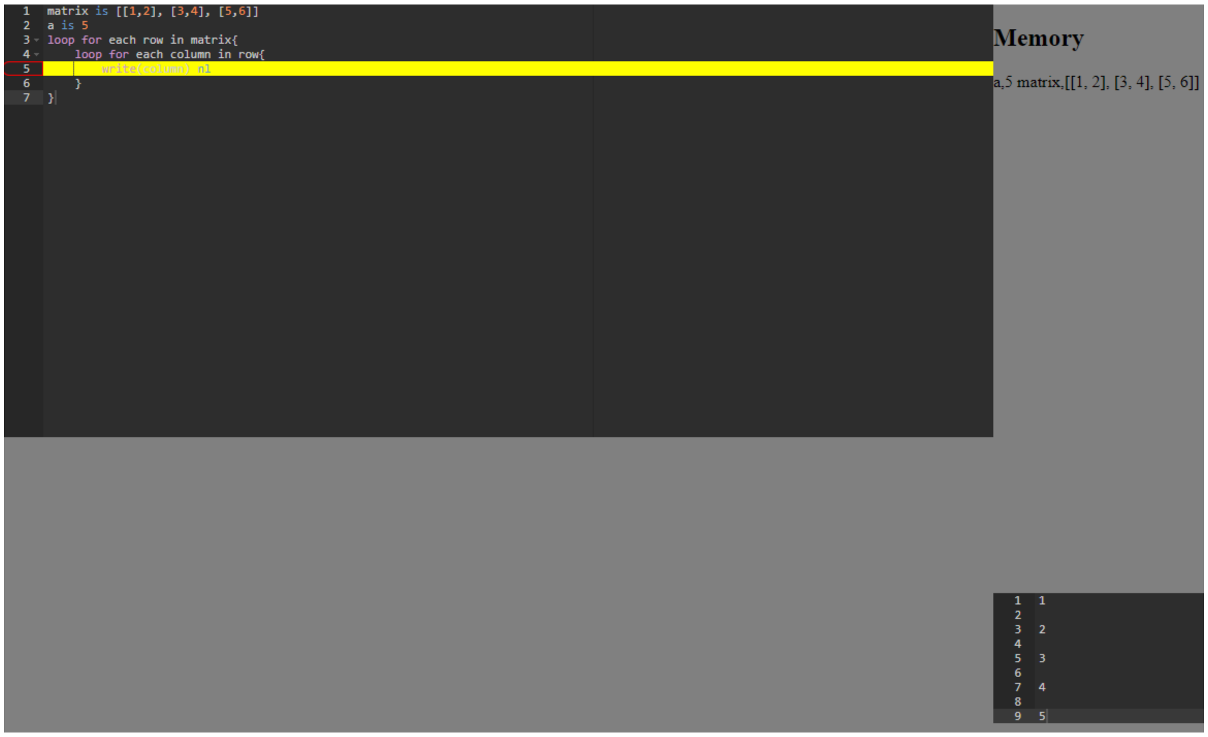


```
matrix is [[1,2], [3,4], [5,6]]
a is 5
loop for each row in matrix{
    loop for each column in row{
        write(column) nl
    }
}
```

Memory

a,5 matrix,[[1, 2], [3, 4], [5, 6]]

```
1

2

3

4
```



```
matrix is [[1,2], [3,4], [5,6]]
a is 5
loop for each row in matrix{
    loop for each column in row{
        write(column) nl
    }
}
```

Memory

a,5 matrix,[[1, 2], [3, 4], [5, 6]]

```
1

2

3

4

5
```

Overall Result

Pass

Test Name : Functions

Test Description

We should be able to define a function, as well as return from one, call a function within a function, and print from a function.

Ensure the breakpoints work as expected.

Test Data

`functions.st`

Expected Result

We expect to see `Hello World` printed when `main` is called correctly, and then see 7 when the `addup` function is called within main and the values passed are returned added together.

There should be (9 points/chosen points) at which the program waits.

Memory updates dynamically.

Execution Steps

Code ran in testing version of START Web Application.

Step button used to advance the program past it's breakpoints.

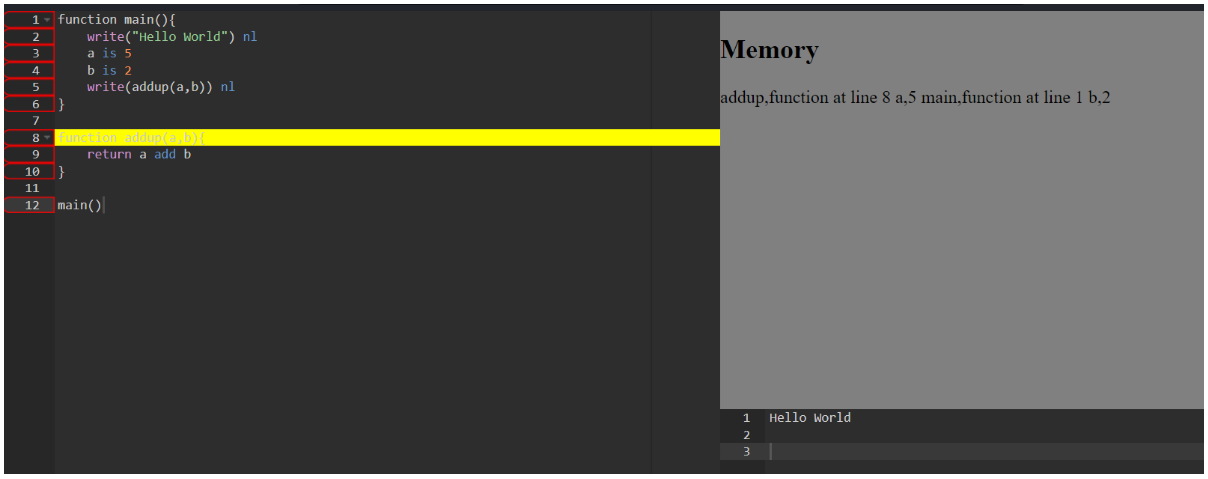
Examine output to ensure variables are of correct value at the end of the program.

Actual Result

The functions were visited in correct order.

- Line-By-Line Mode:
    - Output was correct.
    - There was 9 stopping points.
    - Memory updated dynamically.
- Dynamic Mode:
    - Output was correct.
    - The program stopped the correct number of times.
    - Memory updated dynamically.

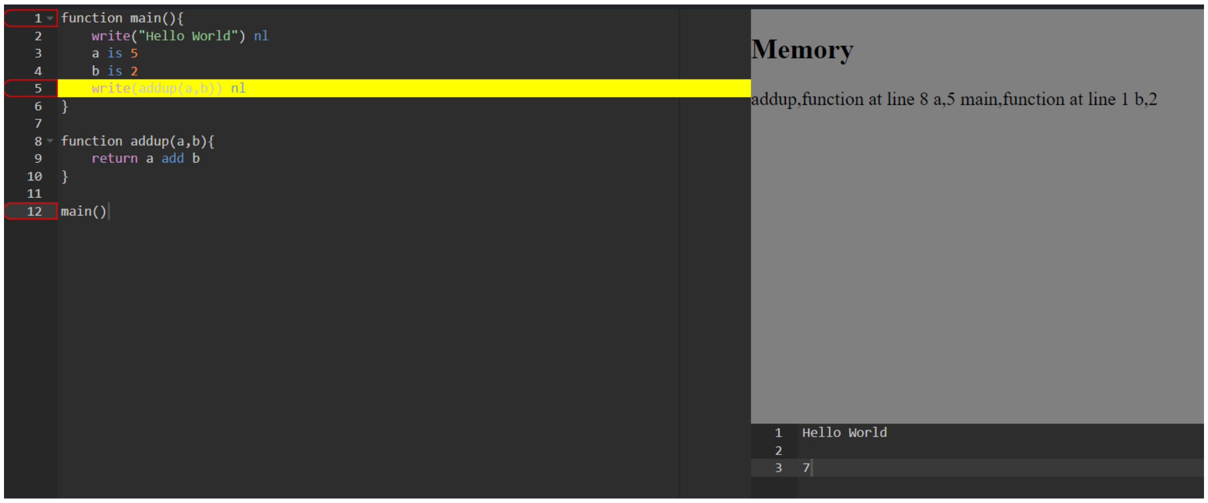
Evidence



```
 1  function main(){
 2      write("Hello World") nl
 3      a is 5
 4      b is 2
 5      write(addup(a,b)) nl
 6  }
 7
 8  function addup(a,b){
 9      return a add b
10  }
11
12  main()
```

addup,function at line 8 a,5 main,function at line 1 b,2

```
1  Hello World
2
3
```



```
1  function main(){
2      write("Hello World") nl
3      a is 5
4      b is 2
5      write(addup(a,b)) nl
6  }
7
8  function addup(a,b){
9      return a add b
10 }
11
12 main()
```

Memory

addup,function at line 8 a,5 main,function at line 1 b,2

```
Hello World

7
```

Overall Result

Pass

Document Review

The initial test revealed 2 major bugs within our code, the 2 tests that contained issues that needed further development time were:

- For Loops
- Functions

Adam was assigned the task of finding and resolving these newly found bugs. Once the bugs had been identified, and subsequently fixed, we needed to then rerun the tests that failed as part of a regression test. Both of these tests then passed the second round of testing. We again found our purpose built test cases were of great use in finding bugs within our code.

Author & Reviewer

Document Author and Tester: Adam Gray

Document and Test Reviewer: Niall Kelly

Date: 03/03/2024