

AMRITA SCHOOL OF COMPUTING CHENNAI
AMRITA VISHWA VIDYAPEETHAM

22AIE113 ELEMENTS OF COMPUTING SYSTEMS-2
ASSESSMENT - 3

DONE BY:
S.PRAVEEN KUMAR
CH.EN.U4AIE22048

1. Write a program in JACK to evaluate $c=a+b$

CODE:

```
class Main{
    function void main() {
        var int a;
        var int b;
        var int c;
        let a = Keyboard.readInt("Enter a:");
        let b = Keyboard.readInt("Enter b:");
        let c = a + b;
        do Output.printInt(c);
        return;
    }
}
```

OUTPUT:

The screenshot displays the JACK IDE interface during program execution. The **Program** window on the left lists 15 instructions, including function calls and arithmetic operations. The **Static**, **Local**, **Argument**, **This**, **That**, and **Temp** windows are currently empty. The **Global Stack** window shows memory addresses from 256 to 270. The **RAM** window shows the state of registers: SP (0), LCL (1), ARG (2), THIS (3), THAT (4), Temp0 (5), Temp1 (6), Temp2 (7), Temp3 (8), Temp4 (9), Temp5 (10), Temp6 (11), Temp7 (12), R13 (13), and R14 (14). The console window on the right shows the input 'Enter a:5', 'Enter b:6', and the output 'Result:11'.

2. Write a Program in JACK to perform linear search.

CODE:

```
class Main{
    function void main(){
        var Array arr;
        var int len;
        var int key;
        var int i;
        var int b ;
        let len = Keyboard.readInt("Enter the len : ");
        let arr = Array.new(len);
        let i = 0;
        while(~(i=len)){
            let arr[i] = Keyboard.readInt("Enter element : ");
            let i = i + 1;
        }

        let key = Keyboard.readInt("Enter the element to search : ");
        let i = 0;
        let b = 0;
        while(~(i = len) & (b = 0)){
            if(arr[i] = key){
                let b = 1;
            }
            let i = i + 1;
        }
        if(b = 0){
            do Output.printString("not found");
        }else{
            do Output.printString("found in index ");
            do Output.printInt(i);
        }
        return;
    }
}
```

OUTPUT:

The screenshot displays a debugger interface with several panels:

- Program:** A list of assembly instructions. Line 121, `if-goto Main.main$WHILE_E...`, is highlighted in yellow.
- Static:** A table for static variables, currently empty.
- Local:** A table for local variables:

Index	Value
0	2059
1	5
2	5
3	-1
4	0
- Argument:** A table for arguments, currently empty.
- This:** A table for the 'this' pointer, currently empty.
- That:** A table for the 'that' pointer:

Index	Value
0	1
1	5
- Temp:** A table for temporary registers:

Index	Value
0	10
1	0
- Global Stack:** A table showing memory addresses and values:

Address	Value
266	2059
267	5
268	5
269	-1
270	0
271	0
272	0
273	5
274	266
275	261
276	0
277	2063
278	5
279	-1
280	278
- Stack:** A single entry with value 0.
- Call Stack:** A list of active functions: `Sys.init (built-in)` and `Main.main` (highlighted in blue).
- RAM:** A table of memory addresses and values:

Address	Value
SP: 0	273
LCL: 1	266
ARG: 2	261
THIS: 3	0
THAT: 4	2059
Temp0: 5	10
Temp1: 6	0
Temp2: 7	0
Temp3: 8	0
Temp4: 9	0
Temp5: 10	0
Temp6: 11	0
Temp7: 12	0
R13: 13	278
R14: 14	115
- Console:** A text area showing program output:

```
Size?5
Enter number:1
Enter number:5
Enter number:6
Enter number:7
Enter number:10
Enter number to find:5
```

3. Write a Program in JACK to perform binary search.

CODE:

```
class Main {
    function void main() {
        var Array arr;
        var int len;
        var int key;
        var int i;
        var int result;
        let len = Keyboard.readInt("Enter the length of the array: ");
        let arr = Array.new(len);
        let i = 0;
        while (i < len) {
            let arr[i] = Keyboard.readInt("Enter element: ");
            let i = i + 1;
        }
        let key = Keyboard.readInt("Enter the element to search: ");

        let result = Main.binary_search(arr, key, 0, len-1);
        if (result = -1) {
            do Output.printString("Element not found.\n");
        } else {
            do Output.printString("Element found at index ");
            do Output.printInt(result);
            do Output.printString("\n");
        }
        return;
    }
    function int binary_search(Array arr, int key, int low, int high){
        var int mid;
        if(low > high){
            return -1;
        }
        let mid = (low + high) / 2;
        if (arr[mid] = key) {
            return mid;
        } else {
            if (arr[mid] < key) {
                return Main.binary_search(arr, key, mid + 1, high);
            } else {
                return Main.binary_search(arr, key, low, mid - 1);
            }
        }
    }
}
```

OUTPUT:

Program

41	push	argument 3
42	call	Main.binary_search 4
43	return	
44	goto	Main.binary_search\$...
	label	Main.binary_search\$...
45	push	argument 0
46	push	argument 1
47	push	argument 2
48	push	local 0
49	push	constant 1
50	sub	
51	call	Main.binary_search 4
52	return	
	label	Main.binary_search\$...
	label	Main.binary_search\$...

Static

0	0
1	0
2	0
3	0
4	0

Local

Argument

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

Enter the length of the array: 3
Enter element: 5
Enter element: 8
Enter element: 1
Enter the element to search: 8
Element found at index 1#

4. Write a recursive program for multiplication.

CODE:

```
class Main {
    function void main () {
        var int a;
        var int b;
        var int c;

        let a = Keyboard.readInt("Enter a :");
        let b = Keyboard.readInt("Enter b :");
        let c = Main.mul(a,b);

        do Output.printString("The product of both = ");
        do Output.printInt(c);

        return;
    }

    function int mul(int a, int b){
        if(b=0){
            return 0;
        }else{
            return a + Main.mul(a,b-1);
        }
    }
}
```

OUTPUT:

Enter First number:15
Enter second number:3
result : 45

Program
101 push constant 32
102 call String.appendChar 2
103 push constant 58
104 call String.appendChar 2
105 push constant 32
106 call String.appendChar 2
107 call Output.printString 1
108 pop temp 0
109 push local 1
110 push local 0
111 call Main.mul 2
112 call Output.printInt 1
113 pop temp 0
114 push constant 0
115 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

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. Write a Program in JACK to perform Matrix Addition.

CODE:

```
class Main {  
  
    function void main() {  
        var Matrix a;  
        var Matrix b;  
        var Matrix c;  
        let a = Matrix.new(2, 2);  
        let b = Matrix.new(2, 2);  
        let c = Matrix.new(2, 2);  
        do Output.printInt(a.getRows());  
        do a.ones();  
        do a.display();  
        do b.ones();  
        do b.display();  
        do a.add(b);  
        do a.display();  
        do Output.printInt(c.index(0));  
        return;  
    }  
}
```

OUTPUT:

The screenshot displays the JACK IDE interface with the following components:

- Program:** A list of instructions for the 'Main.main' function, including 'push', 'call', 'pop', and 'do' statements.
- Static:** A table showing static variables and their values.
- Local:** A table showing local variables and their values.
- Argument:** A table showing argument values.
- This:** A table showing 'this' pointer values.
- That:** A table showing 'that' pointer values.
- Temp:** A table showing temporary variable values.
- Global Stack:** A table showing global stack memory addresses and values.
- RAM:** A table showing RAM memory addresses and values.

Index	Value
0	0
1	0
2	0
3	0
4	0

Index	Value
0	0
1	0
2	0

Index	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0

Index	Value
0	0
1	0
2	0

Index	Value
0	0
1	0

Index	Value
0	0
1	0

Address	Value
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

Address	Value
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

6. Write a Program in JACK to perform factorial of a number using recursion.

CODE:

```
class Main {
    function void main() {
        var int a;
        var int res;

        let a = Keyboard.readInt("Enter the n-th Factorial number: ");
        let res = Main.factorial(a);
        do Output.printString("The n-th Factorial number is ");
        do Output.printInt(res);
        return;
    }

    function int factorial(int n) {
        if (n = 1) {
            return 1;
        }
        return n * Main.factorial(n - 1);
    }
}
```

OUTPUT:

The screenshot displays the JACK IDE interface during the execution of the factorial program. The main window shows the program code with line numbers 44 to 58. The 'Static' memory window shows four slots, all containing 0. The 'Local' memory window is empty. The 'Argument' window is empty. The 'This' window is empty. The 'That' window is empty. The 'Temp' window shows two slots, both containing 0. The 'Stack' window is empty. The 'Call Stack' window is empty. The 'Global Stack' window shows memory addresses 256 to 270, with values 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. The 'RAM' window shows memory addresses 0 to 14, with values 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. The output window shows the text 'Enter n : 6' and 'Result is : 720'.

7. Write a Program in JACK to generate Fibonacci series using recursion.

CODE:

```
class Main {
    function void main() {
        var int a;
        var int res;

        let a = Keyboard.readInt("Enter the n-th Fibonacci number: ");
        let res = Main.fib(a);
        do Output.printString("The n-th Fibonacci number is ");
        do Output.printInt(res);
        return;
    }

    function int fib(int n) {
        if (n = 0) {
            return 0;
        }
        if (n = 1) {
            return 1;
        }
        return Main.fib(n - 1) + Main.fib(n - 2);
    }
}
```

OUTPUT:

The screenshot shows the JACK IDE interface. The Program window on the left displays the source code with line numbers. The Static, Local, Argument, This, That, and Temp windows show memory addresses and values. The Global Stack window shows the stack layout with addresses 256 to 270. The RAM window shows the memory layout with addresses 0 to 14. The output window displays 'result : 13'.

8. Write a program in JACK which generates report card for "n" students.

CODE:

```
class Main {  
  
    function void main() {  
        var Student st;  
        var int sci;  
        var int sc;  
        var int eng;  
        var int maths;  
        var int cs;  
  
        let sci = Keyboard.readInt("Enter the science mark : ");  
        let maths = Keyboard.readInt("Enter the Maths mark : ");  
        let eng = Keyboard.readInt("Enter the English mark : ");  
        let sc = Keyboard.readInt("Enter the Social science mark : ");  
        let cs = Keyboard.readInt("Enter the computer science mark : ");  
  
        let st = Student.new(maths, sci, eng, cs, sc);  
        do st.display();  
  
        return;  
    }  
}
```

OUTPUT:

The screenshot displays the JACK IDE interface. The top toolbar includes navigation icons and settings for animation (Slow, Fast, No animation), view (Screen), and format (Decimal). The main window is divided into several panels:

- Program:** A list of program lines (289-303) showing calls to `String.appendChar 2` and pushes of constant 45.
- Static:** A table showing static memory locations (0-4) with values 0.
- Local:** A table showing local memory locations (0-4) with values 0.
- Argument:** A table showing argument memory locations (0-4) with values 0.
- This:** A table showing 'this' pointer memory locations (0-4) with values 0.
- That:** A table showing 'that' pointer memory locations (0-4) with values 0.
- Temp:** A table showing temporary memory locations (0-1) with values 0.
- Global Stack:** A table showing global stack memory locations (256-270) with values 0.
- RAM:** A table showing RAM memory locations (SP, LCL, ARG, THIS, THAT, Temp0-14, R13, R14) with values 0.

The output window on the right shows the following text:

```
Enter the science mark : 95  
Enter the Maths mark : 97  
Enter the English mark : 98  
Enter the Social science mark : 99  
Enter the computer science mark : 100  
  
-----  
ENGLISH : 98  
Maths : 97  
Science : 95  
Social : 99  
Computer Science : 100  
Percentage: 97  
-----
```

9. Write a jack program which extracts substring from a given sentence/string.

CODE:

```
class Main {
    function void main () {
        var String string;
        var int startingIndex;
        var int endingIndex;
        var String subString;
        let string = Keyboard.readLine("Enter string :");
        let startingIndex = Keyboard.readInt("Enter the starting index :");
        let endingIndex = Keyboard.readInt("Enter the end index :");
        let subString = Main.subString(string, startingIndex, endingIndex);
        do Output.printString("The subString string is ");
        do Output.printString(subString);
        do Output.printString("\n");
        return;
    }
    function String subString (String s, int start, int end ) {
        var String result;
        let result = String.new(end - start);
        while(end > start){
            do result.appendChar(s.charAt(start));
            let start = start + 1;
        }
        return result;
    }
}
```

OUTPUT:

The screenshot displays the Jack IDE interface during the execution of the provided code. The main window shows the following output:

```
Enter string :HELLO HARS
Enter the starting index :0
Enter the end index :10
The subString string is 'HELLO HARS'
```

The IDE also shows the program's execution flow in the 'Program' pane, with line numbers 10 through 23. The 'Static' pane shows memory addresses 0 through 4. The 'Local' pane is empty. The 'Argument' pane is empty. The 'Global Stack' pane shows memory addresses 256 through 270. The 'RAM' pane shows memory addresses 0 through 14.

10. Write a Java program which concatenates 2 strings.

CODE:

```
class Main {
    function void main () {
        var String string1;
        var String string2;
        var String concatenatedString;

        let string1 = Keyboard.readLine("Enter string1 :");
        let string2 = Keyboard.readLine("Enter string2 :");

        let concatenatedString = Main.concat(string1, string2);
        do Output.printString("The concatenated string is ");
        do Output.printString(concatenatedString);
        do Output.printString("\n");
        return;
    }

    function String concat (String a, String b) {
        var int len_b;
        var int len_a;
        var int i;
        var String result;

        let len_b = b.length();
        let len_a = a.length();
        let result = String.new(len_a+len_b);

        let i = 0;
        while(i<len_a){
            do result.appendChar(a.charAt(i));
            let i = i + 1;
        }
        let i = 0;
        while(i<len_b){
            do result.appendChar(b.charAt(i));
            let i = i + 1;
        }

        return result;
    }
}
```

OUTPUT:

The screenshot displays a debugger interface with the following components:

- Program:** A list of instructions from 0 to 14. Instruction 0 is highlighted in yellow.
- Static:** A table for static variables.
- Local:** A table for local variables.
- Argument:** A table for function arguments.
- This:** A table for the 'this' pointer.
- That:** A table for the 'that' pointer.
- Temp:** A table for temporary registers.
- Stack:** A table for the stack.
- Call Stack:** A table showing the call stack with 'Sys.init (built-in)' listed.
- Global Stack:** A table showing memory addresses from 256 to 270.
- RAM:** A table showing memory addresses from 0 to 14.

The output window displays the text "Hello,World?".

Program	function	Main.main 8
0	function	Main.main 8
1	push	constant 6
2	call	String.new 1
3	push	constant 72
4	call	String.appendChar 2
5	push	constant 101
6	call	String.appendChar 2
7	push	constant 108
8	call	String.appendChar 2
9	push	constant 108
10	call	String.appendChar 2
11	push	constant 111
12	call	String.appendChar 2
13	push	constant 44
14	call	String.appendChar 2

Static		

Local		

Argument		

This		

That		

Temp		
0		0
1		0

Global Stack		
256		98
257		0
258		0
259		0
260		0
261		-1
262		261
263		256
264		0
265		0
266		0
267		0
268		0
269		0
270		0

RAM		
SP:	0	266
LCL:	1	266
ARG:	2	261
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	266
R14:	14	-1

11. Write a Program in JACK to perform MERGE sort using recursion.

CODE:

```
class Main {
    function void main () {
        var int len;
        var Array arr;
        var int temp;
        var int i;
        var int j;
        let i = 0;
        let j = 0;
        let len = Keyboard.readInt("Enter the size od array : ");
        let arr = Array.new(len);
        do Output.printInt(len);
        while(~(i=len)){
            let arr[i] = Keyboard.readInt("Enter element : ");
            let i = i + 1;
        }
        let i = 0;
        while(i<(len-1)){
            let j = 0;
            while(j<(len - i - 1)){
                if(arr[j] > arr[j+1]){
                    let temp = arr[j];
                    let arr[j] = arr[j+1];
                    let arr[j+1] = temp;
                }
                let j = j + 1;
            }
            let i = i + 1;
        }
        let i = 0;
        do Output.printString("The sorted array : ");
        do Output.println();
        while(i<len){
            do Output.printInt(arr[i]);
            do Output.printString(", ");
            let i = i + 1;
        }
        do Output.println();
        return;
    }
}
```

```
}  
}
```

OUTPUT:

The screenshot displays a Java IDE interface with the following components:

- Program:** A list of instructions for a program, including calls to `String.appendChar`, `String.appendChar`, `Output.println`, `temp` operations, `push`, `pop`, `add`, `goto`, `label`, `call`, `push`, and `return`.
- Static:** A table showing static variables and their values.
- Local:** A table showing local variables and their values.
- Argument:** A table showing argument variables and their values.
- This:** A table showing the `this` pointer and its value.
- That:** A table showing the `that` pointer and its value.
- Temp:** A table showing temporary variables and their values.
- Global Stack:** A table showing the global stack memory layout, with addresses ranging from 256 to 270.
- RAM:** A table showing the RAM memory layout, with addresses ranging from 0 to 14.
- Output:** A text area showing the program's output, including the size of the array (5) and the sorted array (1, 2, 3, 4, 5).

The output text shows the following sequence of operations:

```
Enter the size of array : 5  
Enter element : 1  
Enter element : 5  
Enter element : 3  
Enter element : 2  
Enter element : 4  
The sorted array :  
1, 2, 3, 4, 5,
```