Department of Computer Science and Engineering
Institute of Technical Education & Research, SOA, Deemed to be University

WEEK-END ASSIGNMENT-06

Pointers and Modular programming

Operating Systems Workshop (CSE 3541)

Problem Statement:

Working with pointers, *referencing* a variable through a pointer and accessing the contents of a memory cell through a pointer variable that stores its address (*i.e. indirect reference*).

Assignment Objectives:

To learn about pointers, referencing, indirect referencing and how to return function results through a function's parameters (input parameters, input/output parameters, output parameters). Also to understand the differences between call-by value & call-by-reference.

Instruction to Students (If any):

Students are required to write his/her own program by avoiding any kind of copy from any sources. Additionally, They must be able to realise the outcome of that question in relevant to systems programming. You may use additional pages on requirement.

Programming/ Output Based Questions:

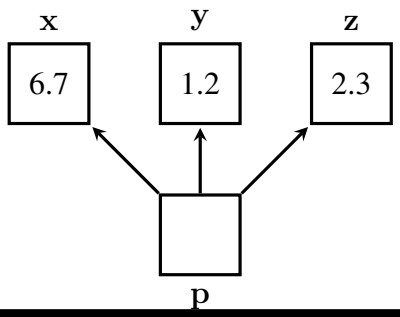
1. For the given structure below, declare the variable type, and print their values as well as addresses;

Space for Program ▼	Output ▼

2. Declare two integer variable and assign values to them, and print their addresses. Additionally, Swap the contents of the variables and print their addresses after swap. State whether the addresses before and after are equal or not.

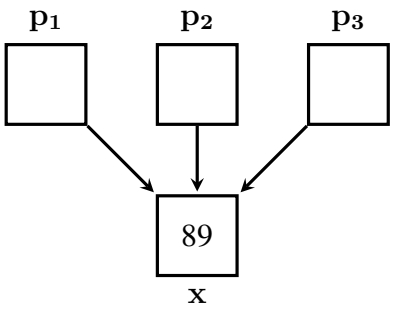
Space for Program ▼	Output ▼

3. Write the C statement to declare and initialize the pointer variable,**p**, for the given structure and display the values of **x**, **y** and **z** with the help of **p**.



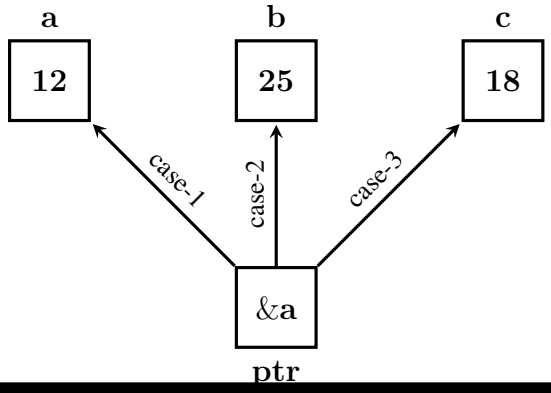
Space for Program ▼ **Output ▼**

4. Write the C statement to declare and initialize the pointer variables p_1, p_2 and p_3 for the given structure and display the value of **x** from p_1. Also update the value of **x** to 100 using pointer p_3.



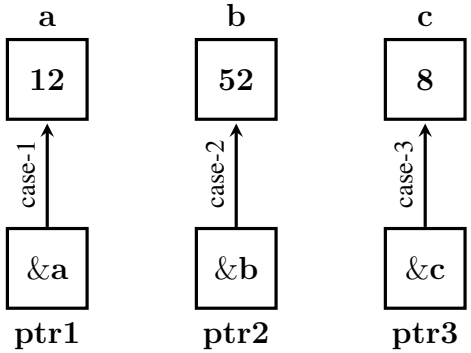
Space for Program ▼ **Output ▼**

5. Write the C statement to declare and initialize the pointer variable for the given structure and update the values of **a**, **b** and **c** to be incremented by 10 through the pointer variable.



Space for Program ▼ **Output ▼**

6. Write the C statement to declare and initialize the pointer variable for the given structure and update the values of **a**, **b** and **c** to be incremented by 10 through their respective pointers.

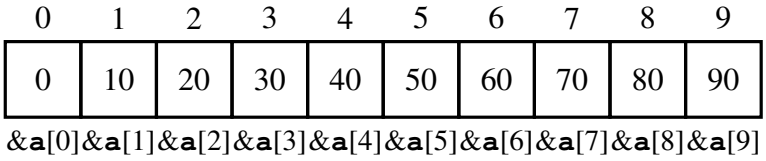


Space for Program ▼ **Output ▼**

7. Two pointers are pointing to different variables. Write the C statement to find the greater between **a**, and **b** using pointer manipulation.

Space for Program ▼ **Output ▼**

8. Create a program to display the address and value of each element of the given integer array **a**. Also perform a close observation on the format of the address and the change of address from index **0** to the last index of the array.



Space for Program ▼ **Output ▼**

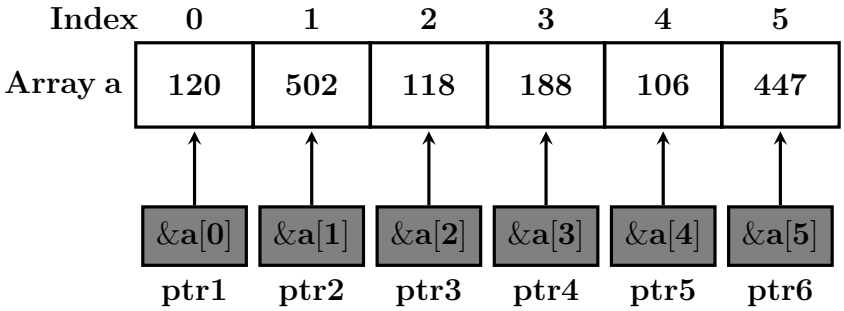
9. Declare the two arrays to hold the values as shown in the given rectangular boxes. Write the equivalent C statement to print their values and addresses through pointer (**`Hint:`** an array name is a pointer to the first element in the array).

10	13	20	33	44

10.2	13.3	20.0	33.3	45.3	89.9

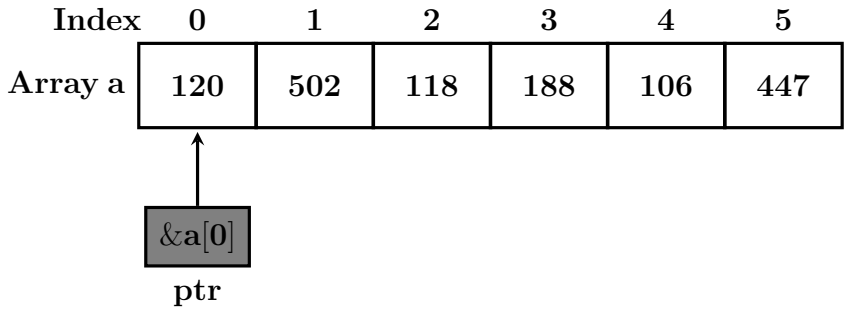
Space for Program ▼ **Output ▼**

10. Write the C statement to declare and initialize the pointer variable for the given structure and display the array content using pointer.



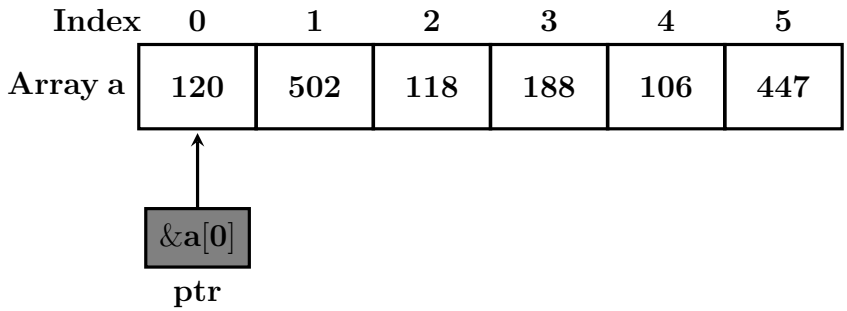
Space for Program **Output ▼**

11. Write the C statement to declare and initialize the pointer variable for the given structure and display the array content using pointer.



Space for Program **Output ▼**

12. As array name is a pointer, so modify the assignment **ptr=a** rather **ptr=&a[0]**. Write the C statement to declare and initialize the pointer variable for the given structure and display the array content using pointer.



Space for Program **Output ▼**

13. Trace the execution of the following fragment at **`line -1`**.

```
int m = 10, n = 5;
int *mp, *np;
mp = &m;
np = &n;
*mp = *mp + *np;
*np = *mp - *np;
printf("%d %d\n%d %d\n", m, *mp, n, *np); /*
    line-1 */
```

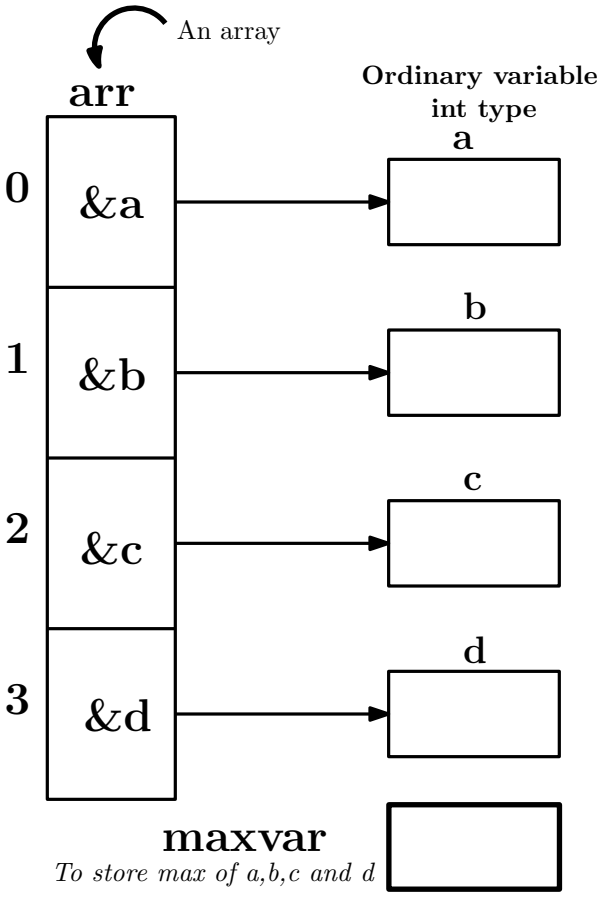
Output▼

14. Given the declarations;

```
int m = 25, n = 77;
char c = '*';
int *itemp;
/* describe the errors in each of the
following statements. */
m = &n;
itemp = m;
*itemp = c;
*itemp = &c;
```

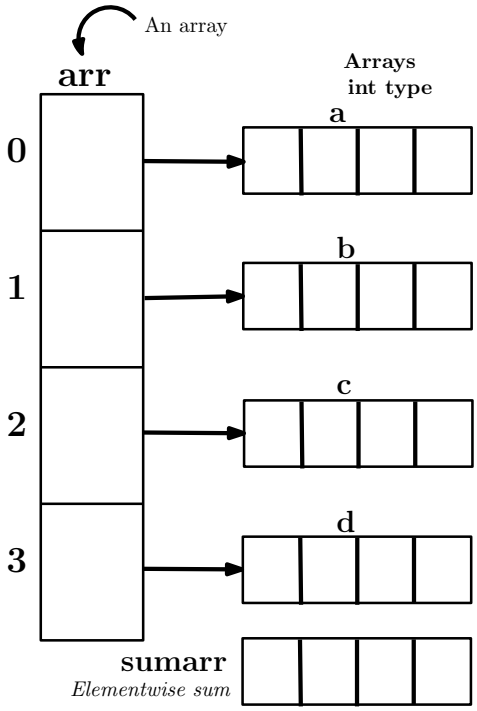
Output▼

15. Simulate the following structure in C to store **55** in **a**, **105** in **b**, **89** in **c** and **68** in **d** using their respective pointers. Additionally find the maximum among **a**, **b**, **c** and **d** through pointer manipulation. Finally Store the maximum to the required variable and display the maximum.



C simulation▼

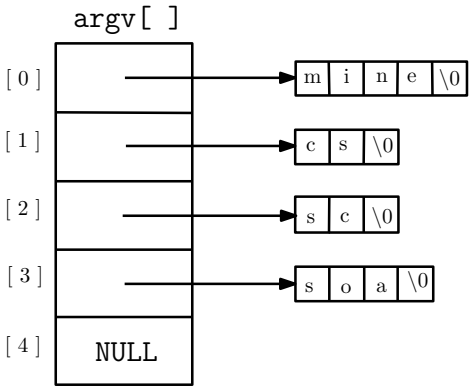
16. Simulate the following structure in C to find the element sum of the given arrays **a**, **b**, **c** and **d** into **sumarray** using their respective pointers. The 1-D arrays must be read/scanned through the pointers.



C simulation▼

Test case: Input & Output▼

17. An argument array is an array of pointers to strings. The end of the array is marked by an entry containing a `NULL` pointer as shown in the figure. Write a C Simulation to implement the following figure and manipuate the character array to hold all capital case letters using pointer. Finally display the strings.



C simulation▼

Test case: Input & Output▼

18. Consider the following figures 1, 2 and 3 to manipulate the ordinary variables, integer arrays and strings through pointers. There exist no names associated with the variables, arrays and strings. State the method to allocate memory for the pointers to manipulate the desired variables.

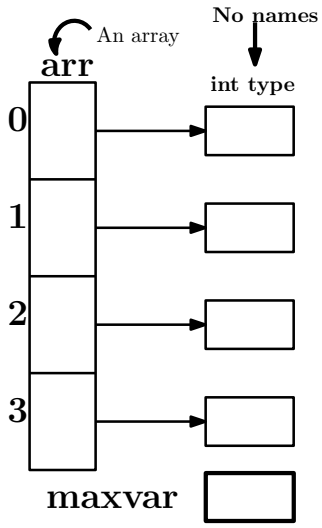


Figure-1

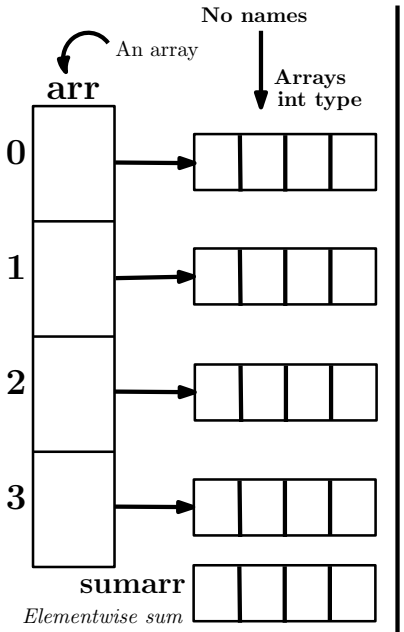


Figure-2

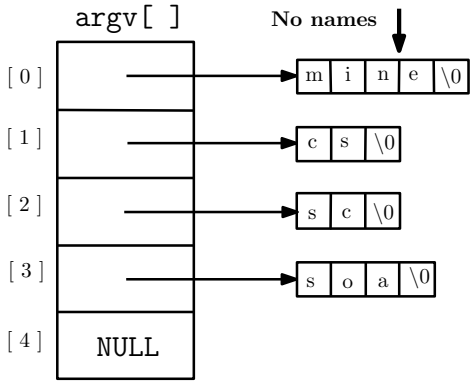
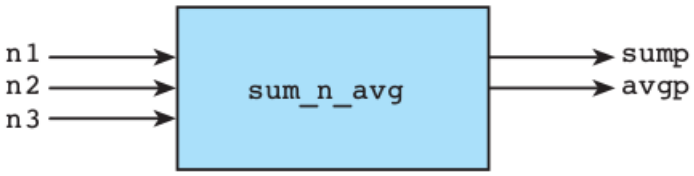


Figure-3

Answer ▼

19. Write a prototype for a function **sum_n_avg** that has three type double input parameters and two output parameters. The function computes the sum and the average of its three input arguments and relays its results through two output parameters.



Output▼

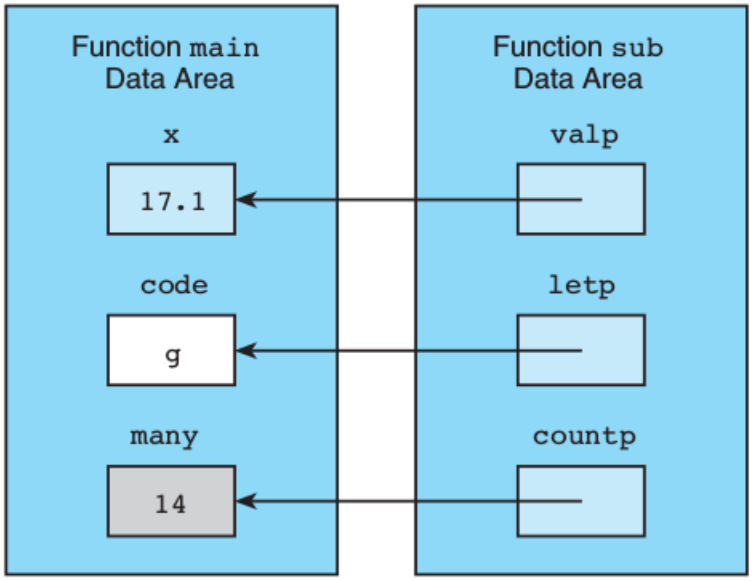
20. The following code fragment is from a function preparing to call **sum_n_avg** (see question-19). Complete the function call statement.

```
double one, two, three, sum_of_3, avg_of_3;
printf("Enter three numbers> ");
scanf("%lf%lf%lf", &one, &two, &three);
sum_n_avg(_____________________________________________________________);
. . .
```

Define the function **sum_n_avg** whose prototype you wrote in question-19.

Space for Program	Output ▼

21. Given the memory setup shown, fill in the chart by indicating the data type and value of each reference as well as the name of the function in which the reference would be legal. Describe pointer values by



referring to cell attributes. For example, the value of **`valp`** would be "pointer to color-shaded cell", and the value of **`&many`** would be "pointer to gray-shaded cell".

Reference	Where Legal	Data Type	Value
`valp`	`sub`	`double *`	pointer to color-shaded cell
`&many`			
`code`			
`&code`			
`countp`			
`*countp`			
`*valp`			
`letp`			
`&x`			

22. Write a program to use the idea of **multiple calls to a function with input/output parameters** to sort 6 integer numbers in ascending order without using any sorting algorithms. The prototype of the function to be used in your program to sort the numbers is given as **`void arrange(int *, int *);`** and also draw the data areas of calling function and **`arragne()`** function for the first function call **`arrange(....)`**.

Sample Run

```
printf("Enter SIX numbers separated by blanks> ");
12 3 56 8 20 654

/* Displays results */
printf("The numbers in ascending order are: %d %d %d %d %d %d\n", n1,
    n2, n3, n4, n5, n6);
3 8 12 20 56 654
```

Space for Program ▼ **Output ▼**

23. Show the table of values for x , y , and z that is the output displayed by the following program.

```
#include <stdio.h>
void sum(int a, int b, int *cp);
int main(void){
 int x, y, z;
 x = 7; y = 2;
 printf("x y z\n\n");
 sum(x, y, &z);
 printf("%4d%4d%4d\n", x, y, z);
 sum(y, x, &z);
 printf("%4d%4d%4d\n", x, y, z);
 sum(z, y, &x);
 printf("%4d%4d%4d\n", x, y, z);
 sum(z, z, &x);
 printf("%4d%4d%4d\n", x, y, z);
 sum(y, y, &y);
 printf("%4d%4d%4d\n", x, y, z);
 return (0);
}
```

```
void sum(int a, int b, int *cp){
  *cp = a + b;
}
```

Output▼

24. (a) What values of x and y are displayed by this program? (Hint: Sketch the data areas of **`main`** , **`trouble`** , and **`double_trouble`** as the program executes.)

```
void double_trouble(int *p, int y);
void trouble(int *x, int *y);
int main(void){
  int x, y;
  trouble(&x, &y);
  printf("x = %d, y = %d\n", x, y);
  return (0);
}
void double_trouble(int *p, int y){
  int x;
  x = 10;
  *p = 2 * x - y;
}
void trouble(int *x, int *y){
  double_trouble(x, 7);
  double_trouble(y, *x);
}
```

Output▼

(b) Classify each formal parameter of **`double_trouble`** and **`trouble`** as input, output, or input/output.

Formal parameter classification ▼

25. A finite state machine (FSM) consists of a set of states, a set of transitions, and a string of input data. In the FSM of Figure 1, the named ovals represent states, and the arrows connecting the states represent transitions. The FSM is designed to recognize a list of **C identifiers** and **nonnegative integers**, assuming that the items are ended by one or more blanks and that a period marks the end of all the data. The following table traces how the diagrammed machine would process a string composed of one blank, the digits 9 and 5, two blanks, the letter K, the digit 9, one blank, and a period. The machine begins in the start state.

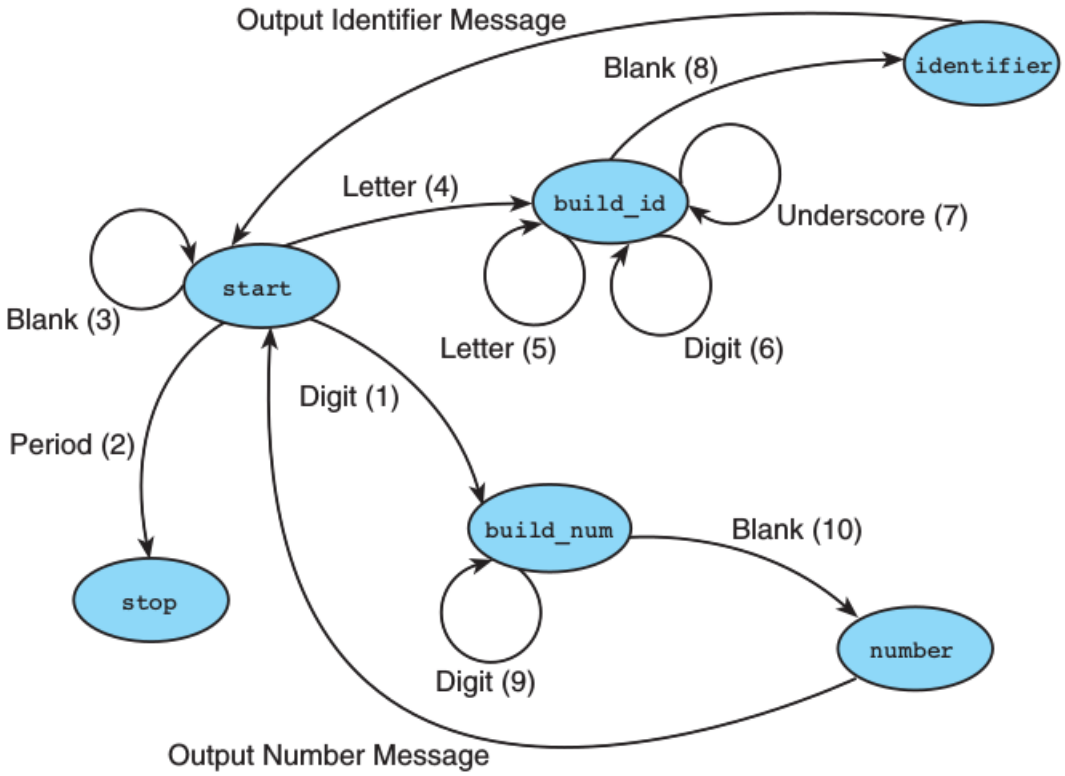


Figure 1: Finite State Machine for Numbers and Indentifiers

Figure 2: **FSM tracing for 95 and K9**

State	Next Character	Transition
start	' '	3
start	'9'	1
build_num	'5'	9
build_num	' '	10
number		Output number message
start	' '	3
start	'K'	4
build_id	'9'	6
build_id	' '	8
identifier		Output identifier message
start	'.'	2
stop		

Write a program that uses an enumerated type to represent the names of the states. Your program should process a correctly formatted line of data, identifying each data item. Here is a sample of correct input and output.

```
Input :

rate R2D2 48 2 time 555666

Output :

rate - Identifier
R2D2 - Identifier
48 - Number
2 - Number
time - Identifier
555666 - Number
```

Use the following code fragment in **`main`** , and design function **`transition`** to return the next state for all the numbered transitions of the finite state machine. If you include the header file **`ctype.h`**, you can use the library function **`isdigit`** which returns 1 if called with a digit character, 0 otherwise. Similarly, the function **`isalpha`** checks whether a character is a let- ter. When your program correctly models the behavior of the **`FSM`** shown, extend the **`FSM`** and your program to allow optional signs and optional fractional parts (i.e., a decimal point followed by zero or more digits) in numbers.

```
current_state = start;
do {
   if (current_state == identifier) {
        printf(" - Identifier\n");
        current_state = start;
   } else if (current_state == number) {
        printf(" - Number\n");
        current_state = start;
   }
   scanf("%c", &transition_char);
   if (transition_char != ' ')
        printf("%c", transition_char);
        current_state = transition(current_state, transition_char);
   } while (current_state != stop);
```

FSM Inplementation ▼

FSM Inplementation ▼

FSM Inplementation ▼