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:

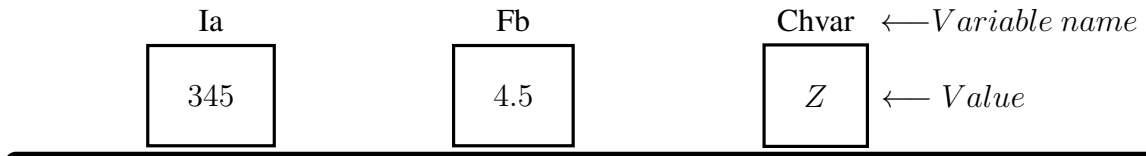
To learn about pointers, referencing, indirect referencing and how to return function results through a function's parameters (input parameters, input/output parameters, output parameters). Also to understand the differences between call-by value & call-by-reference.

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 ▼

```
#include <stdio.h>

int main() {
   int Ia=345;
   float Fb=4.5;
   char chvar='Z';
   printf("Ia: %d, address: %p\n", Ia, (void*)&Ia);
   printf("Fb: %f, address: %p\n", Fb, (void*)&Fb);
   printf("chvar: %c, address: %p\n", chvar, (void*)&chvar);

   return 0;
}
```

Output ▼

```
Ia: 345, address: 0x7fff9ac0e700
Fb: 4.500000, address: 0x7fff9ac0e704
chvar: Z, address: 0x7fff9ac0e6ff
```

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.

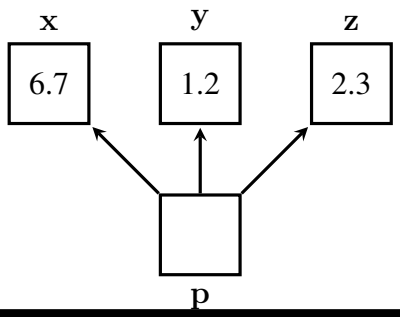
Space for Program ▼

```
#include <stdio.h>
int main() {
   int num1, num2;
   num1 = 10;
   num2 = 20;
   printf("Address of num1 before swap: %p\n", (void*)&num1);
   printf("Address of num2 before swap: %p\n", (void*)&num2);
   int *ptr1 = &num1;
   int *ptr2 = &num2;
   int temp = *ptr1;
   *ptr1 = *ptr2;
   *ptr2 = temp;
   printf("Address of num1 after swap: %p\n", (void*)&num1);
   printf("Address of num2 after swap: %p\n", (void*)&num2);
   if (&num1 == &num2) {
      printf("Addresses are equal before and after swap.\n");
   } else {
      printf("Addresses are not equal before and after swap.\n");
   }
   return 0;
}
```

Output ▼

```
Address of num1 before swap: 0x7ffe7bf3359c
Address of num2 before swap: 0x7ffe7bf335a0
Address of num1 after swap: 0x7ffe7bf3359c
Address of num2 after swap: 0x7ffe7bf335a0
Addresses are not equal before and after swap.
```

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**.

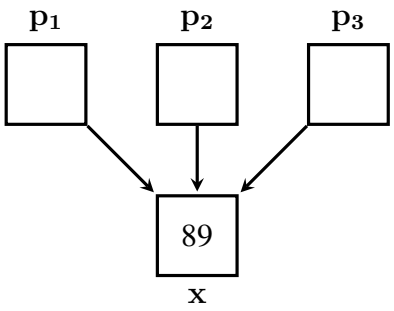


Space for Program ▼ **Output ▼**

```
#include <stdio.h>

int main() {
   void *p;
   float x = 6.7;
   float y = 1.2;
   float z = 2.3;
   p = &x;
   printf("Value of x: %.1f\n", *(float*)p);
   p = &y;
   printf("Value of y: %.1f\n", *(float*)p);
   p = &z;
   printf("Value of z: %.1f\n", *(float*)p);
   return 0;
}
```

```
Value of x: 6.7

Value of y: 1.2

Value of z: 2.3
```

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.

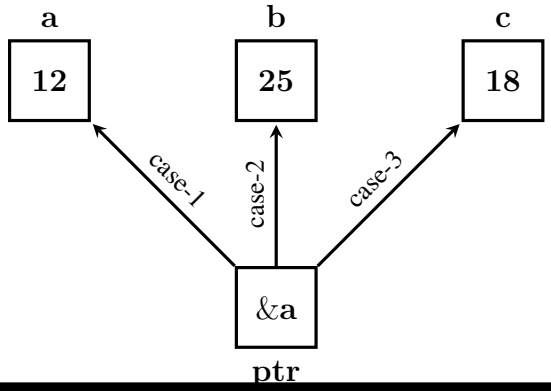


Space for Program ▼ **Output ▼**

```
#include <stdio.h>

int main() {
   int x = 89;
   int *p1 = &x;
   int *p2 = &x;
   int *p3 = &x;
   printf("Value of x from p1: %d\n", *p1);
   printf("Value of x from p2: %d\n", *p2);
   printf("Value of x from p2: %d\n", *p3);
   *p3 = 100;
   printf("Updated value of x from p3: %d\n", *p3);

   return 0;
}
```

```
Value of x from p1: 89

Value of x from p2: 89

Value of x from p2: 89

Updated value of x from p3: 100
```

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.

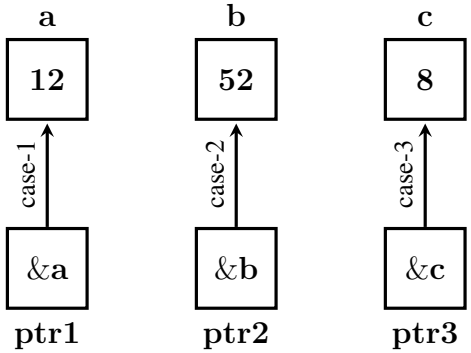


Space for Program ▼ **Output ▼**

```
#include <stdio.h>
int main() {
   int a = 12;
   int b = 25;
   int c = 18;
   int *ptr;
   ptr = &a;
   *ptr=*ptr+10;
   ptr = &b;
   *ptr=*ptr+10;
   ptr = &c;
   *ptr=*ptr+10;
   printf("Updated value of a: %d\n", a);
   printf("Updated value of b: %d\n", b);
   printf("Updated value of c: %d\n", c);
   return 0;
}
```

```
Updated value of a: 22
Updated value of b: 35
Updated value of c: 28
```

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 ▼**

```
#include <stdio.h>

int main() {
   int a = 12;
   int b = 25;
   int c = 18;
   int *ptr=&a;
   *ptr=*ptr+10;
   int *ptr1 = &b;
   *ptr1=*ptr1+10;
   int *ptr2 = &c;
   *ptr2=*ptr2+10;
   printf("Updated value of a: %d\n", a);
   printf("Updated value of b: %d\n", b);
   printf("Updated value of c: %d\n", c);
   return 0;
}
```

```
Updated value of a: 22
Updated value of b: 35
Updated value of c: 28
```

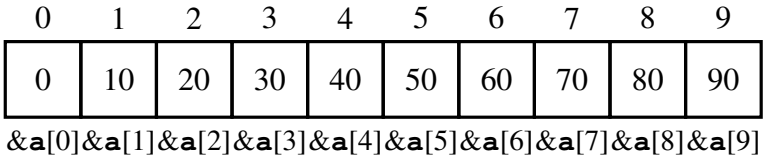
7. Two pointers are pointing to different variables. Write the C statement to find the greater between **a**, and **b** using pointer manipulation.

Space for Program ▼ **Output ▼**

```
#include <stdio.h>

int main() {
   int a = 52;
   int b = 18;
   int *ptrA = &a;
   int *ptrB = &b;
   if (*ptrA > *ptrB) {
      printf("The greater value is: %d\n", *ptrA);
   } else if (*ptrA < *ptrB) {
      printf("The greater value is: %d\n", *ptrB);
   } else {
      printf("Both values are equal: %d\n", *ptrA);
   }
   return 0;
}
```

```
The greater value is: 52
```

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 ▼**

```
#include <stdio.h>

int main() {
   int a[] = {0, 10, 20, 30, 40, 50, 60, 70, 80, 90};
   int arraySize = sizeof(a) / sizeof(a[0]);
   printf("Array Elements:\n");
   for (int i = 0; i < arraySize; ++i) {
      printf("Index %d - Address: %p, Value: %d\n", i, (a + i), *(a + i));
   }
   printf("\nObservation of Address Change:\n");
   int *ptr = a;
   for (int i = 0; i < arraySize; ++i) {
      printf("Index %d - Address: %p\n", i, (ptr + i));
   }

   return 0;
}
```

```
Array Elements:
Index 0 - Address: 0x7ffd4ca75050, Value: 0
Index 1 - Address: 0x7ffd4ca75054, Value: 10
Index 2 - Address: 0x7ffd4ca75058, Value: 20
Index 3 - Address: 0x7ffd4ca7505c, Value: 30
Index 4 - Address: 0x7ffd4ca75060, Value: 40
Index 5 - Address: 0x7ffd4ca75064, Value: 50
Index 6 - Address: 0x7ffd4ca75068, Value: 60
Index 7 - Address: 0x7ffd4ca7506c, Value: 70
Index 8 - Address: 0x7ffd4ca75070, Value: 80
Index 9 - Address: 0x7ffd4ca75074, Value: 90

Observation of Address Change:
Index 0 - Address: 0x7ffd4ca75050
Index 1 - Address: 0x7ffd4ca75054
Index 2 - Address: 0x7ffd4ca75058
Index 3 - Address: 0x7ffd4ca7505c
Index 4 - Address: 0x7ffd4ca75060
Index 5 - Address: 0x7ffd4ca75064
Index 6 - Address: 0x7ffd4ca75068
Index 7 - Address: 0x7ffd4ca7506c
Index 8 - Address: 0x7ffd4ca75070
Index 9 - Address: 0x7ffd4ca75074
```

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

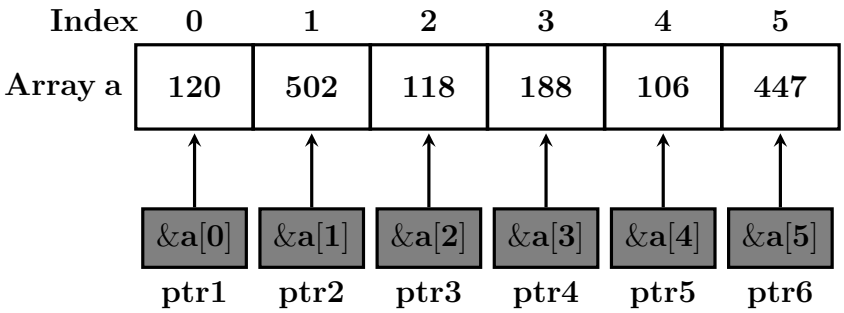
Space for Program ▼

```
#include <stdio.h>

int main() {
   int array1[] = {10,13,20,33,44};
   float array2[] = {10.2,13.3,20.2,33.3,45.3,89.9};
   printf("Array1 Elements:\n");
   int *ptr1 = array1;
   for (int i = 0; i < sizeof(array1) / sizeof(array1[0]); i++) {
      printf("Index %d - Address: %p, Value: %d\n", i, (ptr1 + i), *(ptr1 + i));
   }
   printf("\nArray2 Elements:\n");
   float *ptr2 = array2;
   for (int i = 0; i < sizeof(array2) / sizeof(array2[0]); ++i) {
      printf("Index %d - Address: %p, Value: %0.1f\n", i, (ptr2 + i), *(ptr2 + i));
   }
   return 0;
}
```

Output ▼

```
Array1 Elements:
Index 0 - Address: 0x7ffd30465d60, Value: 10
Index 1 - Address: 0x7ffd30465d64, Value: 13
Index 2 - Address: 0x7ffd30465d68, Value: 20
Index 3 - Address: 0x7ffd30465d6c, Value: 33
Index 4 - Address: 0x7ffd30465d70, Value: 44

Array2 Elements:
Index 0 - Address: 0x7ffd30465d80, Value: 10.2
Index 1 - Address: 0x7ffd30465d84, Value: 13.3
Index 2 - Address: 0x7ffd30465d88, Value: 20.2
Index 3 - Address: 0x7ffd30465d8c, Value: 33.3
Index 4 - Address: 0x7ffd30465d90, Value: 45.3
Index 5 - Address: 0x7ffd30465d94, Value: 89.9
```

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



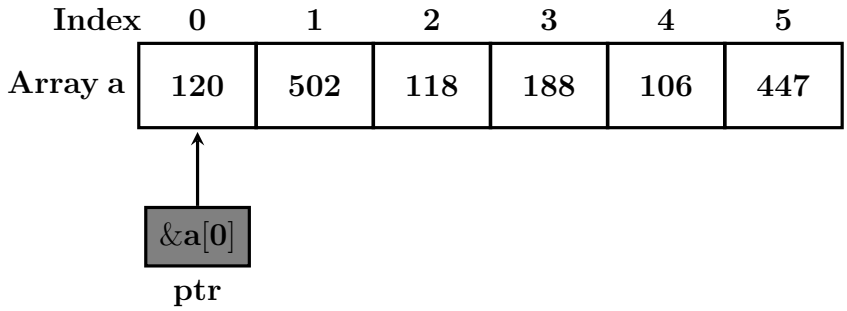
Space for Program

```
#include <stdio.h>

int main() {
   int a[] = {120, 502, 118, 188, 106, 447};
   int *ptr1= a;
   int *ptr2 = a + 1;
   int *ptr3 = a + 2;
   int *ptr4 = a + 3;
   int *ptr5 = a + 4;
   int *ptr6 = a + 5;
   printf("Array Content using Pointer:\n");
   for (int i = 0; i < sizeof(a) / sizeof(a[0]); i++) {
      printf("Index %d - Value: %d\n", i, *(ptr1 + i));
   }
   return 0;
}
```

Output ▼

```
Array Content using Pointer:

Index 0 - Value: 120

Index 1 - Value: 502

Index 2 - Value: 118

Index 3 - Value: 188

Index 4 - Value: 106

Index 5 - Value: 447
```

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

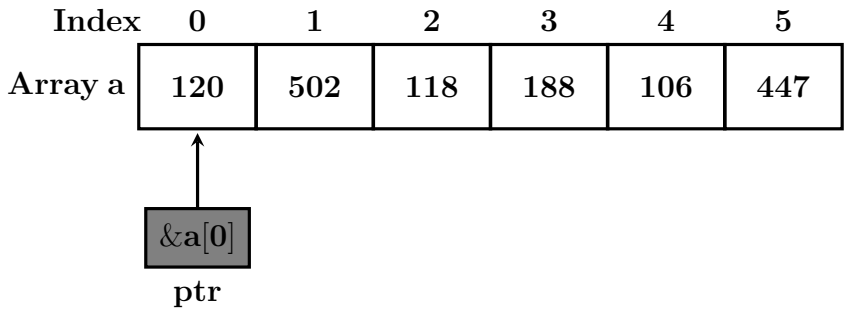


Space for Program **Output ▼**

```
#include <stdio.h>

int main() {
   int myArray[] = {120, 502, 118, 188, 106, 447};
   int *ptr = myArray;
   printf("Array Content using Pointer:\n");
   for (int i = 0; i < sizeof(myArray) / sizeof(myArray[0]); i++) {
      printf("Index %d - Value: %d\n", i, *(ptr + i));
   }
   return 0;
}
```

```
Array Content using Pointer:
Index 0 - Value: 120
Index 1 - Value: 502
Index 2 - Value: 118
Index 3 - Value: 188
Index 4 - Value: 106
Index 5 - Value: 447
```

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 ▼**

```
#include <stdio.h>

int main() {
   int a[] = {120, 502, 118, 188, 106, 447};
   int *ptr = a;
   printf("Array Content using Pointer:\n");
   for (int i = 0; i < sizeof(a) / sizeof(a[0]); i++) {
      printf("Index %d - Value: %d\n", i, *(ptr + i));
   }
   return 0;
}
```

```
Array Content using Pointer:
Index 0 - Value: 120
Index 1 - Value: 502
Index 2 - Value: 118
Index 3 - Value: 188
Index 4 - Value: 106
Index 5 - Value: 447
```

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▼

```
15 15
10 10
```

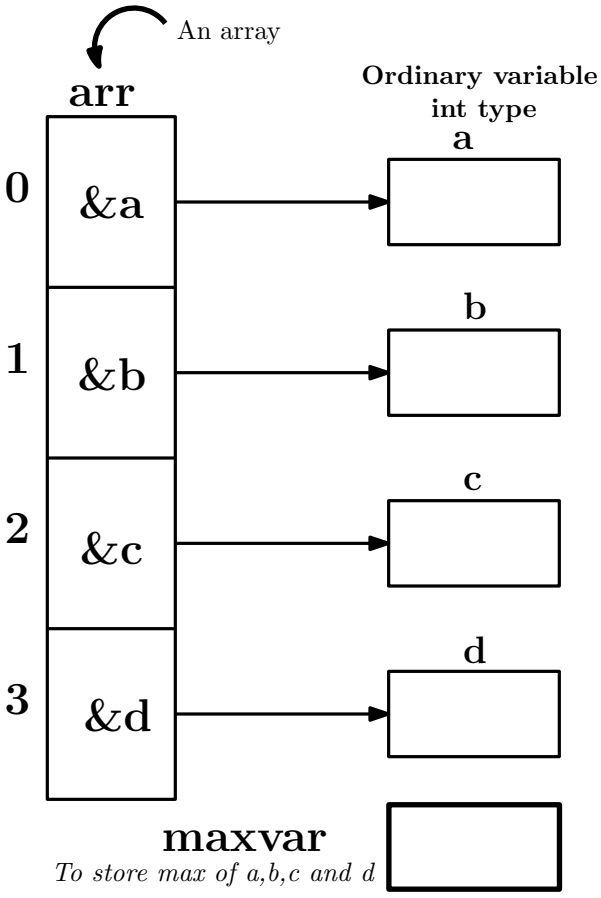
14. Given the declarations;

```
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▼

```
Trying to store pointer value
in integer variable
```

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.



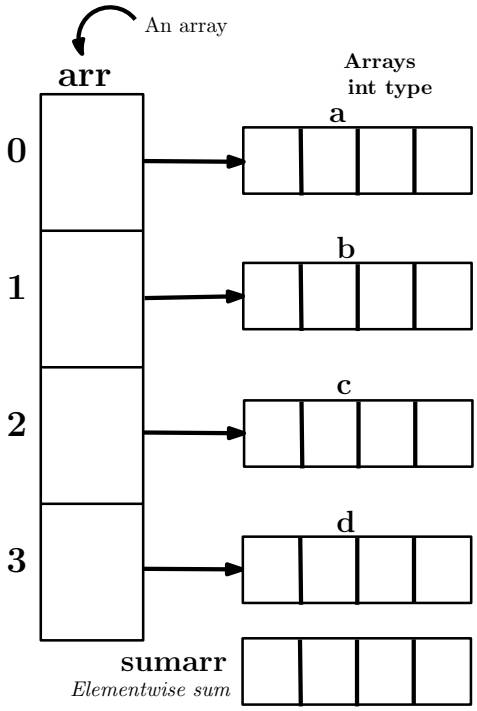
C simulation▼

```
#include <stdio.h>

int main() {
   int a, b, c, d;
   int *ptrA = &a;
   int *ptrB = &b;
   int *ptrC = &c;
   int *ptrD = &d;
   *ptrA = 55;
   *ptrB = 105;
   *ptrC = 89;
   *ptrD = 68;
   int *maxPtr = ptrA;
   if (*ptrB > *maxPtr) {
      maxPtr = ptrB;
   }
   if (*ptrC > *maxPtr) {
      maxPtr = ptrC;
   }
   if (*ptrD > *maxPtr) {
      maxPtr = ptrD;
   }
   int maximumValue = *maxPtr;
   printf("Maximum value: %d\n", maximumValue);

   return 0;
}
```

OUTPUT:-

```
Maximum value: 105
```

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.



Test case: Input & Output▼

Sum of a: 150

Sum of b: 175

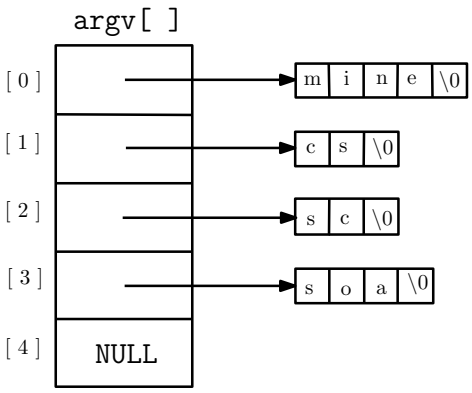
Sum of c: 75

Sum of d: 105

C simulation▼

```
#include <stdio.h>

int main() {

    int a[] = {10, 20, 30, 40, 50};
    int b[] = {15, 25, 35, 45, 55};
    int c[] = {5, 10, 15, 20, 25};
    int d[] = {7, 14, 21, 28, 35};

    int *ptr_a = a;
    int *ptr_b = b;
    int *ptr_c = c;
    int *ptr_d = d;

    int sum_a = 0, sum_b = 0, sum_c = 0, sum_d = 0;
    for (int i = 0; i < 5; i++) {
        sum_a += *(ptr_a + i);
        sum_b += *(ptr_b + i);
        sum_c += *(ptr_c + i);
        sum_d += *(ptr_d + i);
    }
    int sumarray[] = {sum_a, sum_b, sum_c, sum_d};

    printf("Sum of a: %d\n", sum_a);
    printf("Sum of b: %d\n", sum_b);
    printf("Sum of c: %d\n", sum_c);
    printf("Sum of d: %d\n", sum_d);

    return 0;
}
```

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▼

```
#include <stdio.h>

void convertToUpperCase(char *str) {
   while (*str != '\0') {
      if (*str >= 'a' && *str <= 'z') {
         *str = *str - ('a' - 'A');
      }
      str++;
   }
}
int main() {
   char arguments[ ][20] = {"mine", "cs", "sc", "soa"};
   for (int i = 0; i < sizeof(arguments) / sizeof(arguments[0]); i++) {
      convertToUpperCase(arguments[i]);
   }
   for (int i = 0; i < sizeof(arguments) / sizeof(arguments[0]); i++) {
      printf("%s\n", arguments[i]);
   }
   return 0;
}
```

Test case: Input & Output▼

MINE

CS

SC

SOA

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.

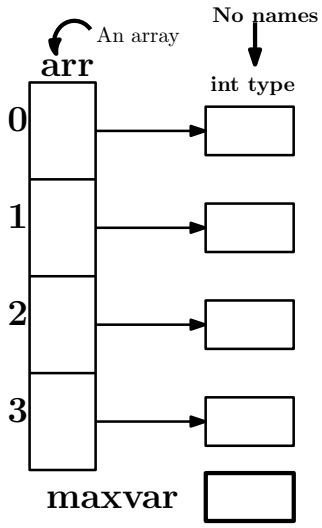


Figure-1

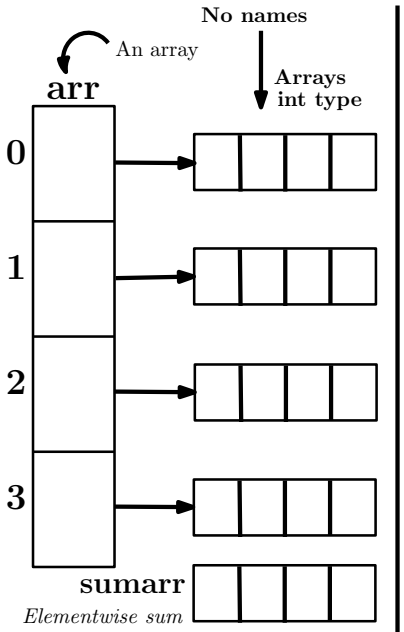


Figure-2

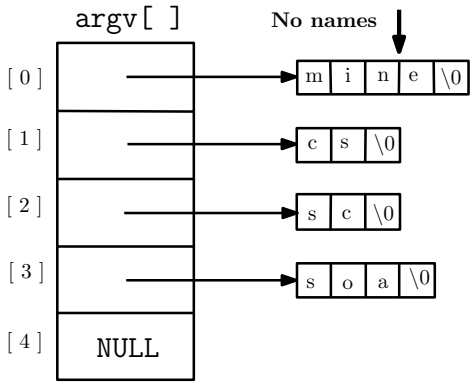


Figure-3

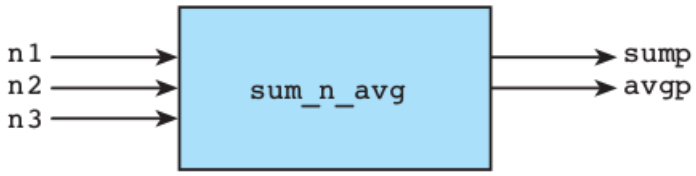
Answer ▼

a.) int *ptr;

b.) int *arr = (int*)malloc(5*sizeof(int));

c.) char *str = (char*)malloc(50*sizeof(char));

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▼

void sum_n_avg(double num1,double num2
double num3,double *sum, double *avg);

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

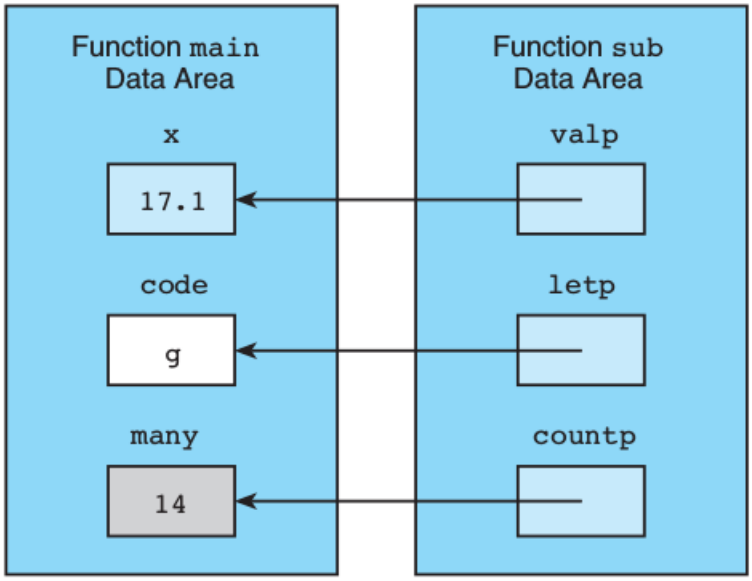
```
double one, two, three, sum_of_3, avg_of_3;
printf("Enter three numbers> ");
scanf("%lf%lf%lf", &one, &two, &three);
sum_n_avg(_____);
. . .
```

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

Space for Program	Output ▼

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

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 ▼**

```
#include <stdio.h>

void arrange(int *a, int *b) {
    if (*a > *b) {
        int temp = *a;
        *a = *b;
        *b = temp;
    }
}

int main() {
    int n1, n2, n3, n4, n5, n6;

    printf("Enter SIX numbers separated by blanks: ");
    scanf("%d %d %d %d %d %d", &n1, &n2, &n3, &n4, &n5, &n6);

    arrange(&n1, &n2);
    arrange(&n1, &n3);
    arrange(&n1, &n4);
    arrange(&n1, &n5);
    arrange(&n1, &n6);

    arrange(&n2, &n3);
    arrange(&n2, &n4);
    arrange(&n2, &n5);
    arrange(&n2, &n6);

    arrange(&n3, &n4);
    arrange(&n3, &n5);
    arrange(&n3, &n6);

    arrange(&n4, &n5);
    arrange(&n4, &n6);

    arrange(&n5, &n6);

    printf("The numbers in ascending order are: %d %d %d %d %d %d\n", n1, n2, n3, n4, n5, n6);

    return 0;
}
```

```
Enter SIX numbers separated by blanks: 10 20 5 2 1 3

The numbers in ascending order are: 1 2 3 5 10 20
```

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▼

```
x y z

  7   2   0
  7   2   9
  7   2   9
 11   2   9
 18   2   9
 18   4   9
```

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▼

```
x = 13, y = 7
```

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

Formal parameter classification ▼

```
double_trouble='y'

trouble='x' & 'y'
```

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.

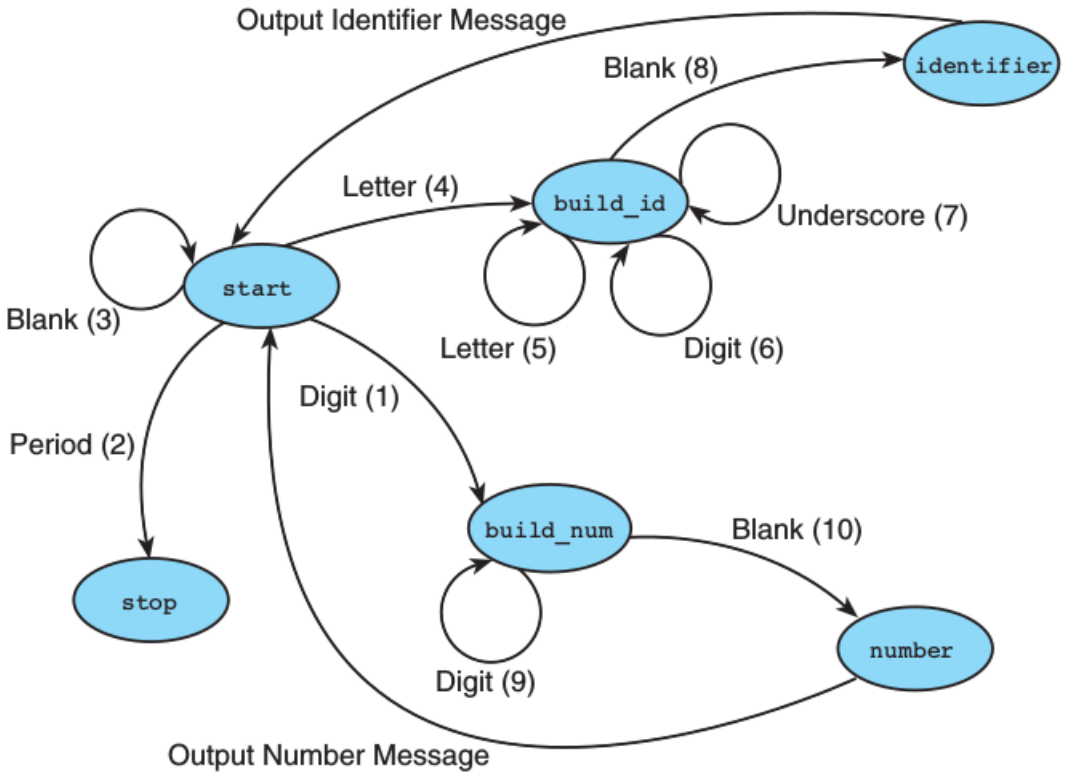


Figure 1: Finite State Machine for Numbers and Indentifiers

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 ▼

```
#include <stdio.h>
#include <ctype.h>
enum State {
   start,identifier,number,stop};
enum State transition(enum State current, char c) {
   if (current == start) {
      if (isalpha(c)) {
         return identifier;
      } else if (isdigit(c)) {
         return number;}
   } else if (current == identifier) {
      if (isalnum(c) || c == '_') {
         return identifier;}
   } else if (current == number) {
      if (isdigit(c) || c == '.') {
         return number;}}
   return stop;
}
int main() {
   enum State current_state = start;
   char transition_char;
   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);
   return 0;
}
```

OUTPUT:-

*** rate-Identifier**

*** R2D2-Identifier**

*** 48-Number**

*** 2- Number**

*** time-Identifier**

*** 555666-Number**

rate R2D2 48 2 time 555666