KHUSHI MALIK

500119018

R2142230031

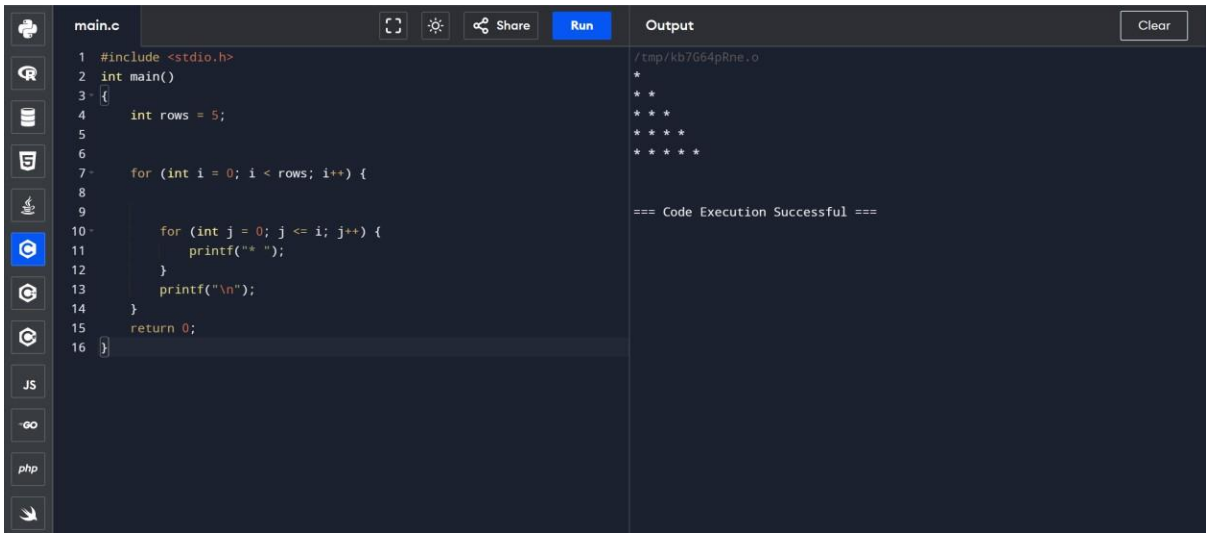
BATCH -2, SEM-3

DESIGN AND ANALYSIS OF ALGORITHMS (DAA)

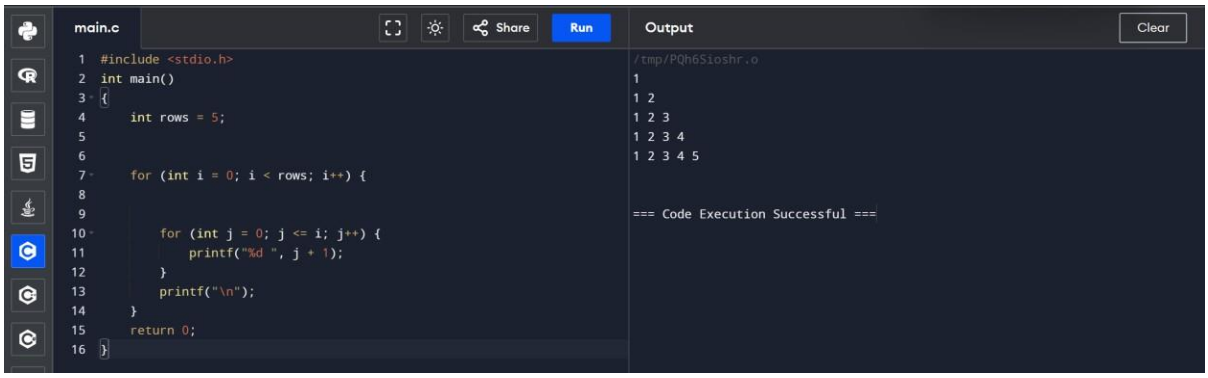
ALL LABS

- PATTERNS

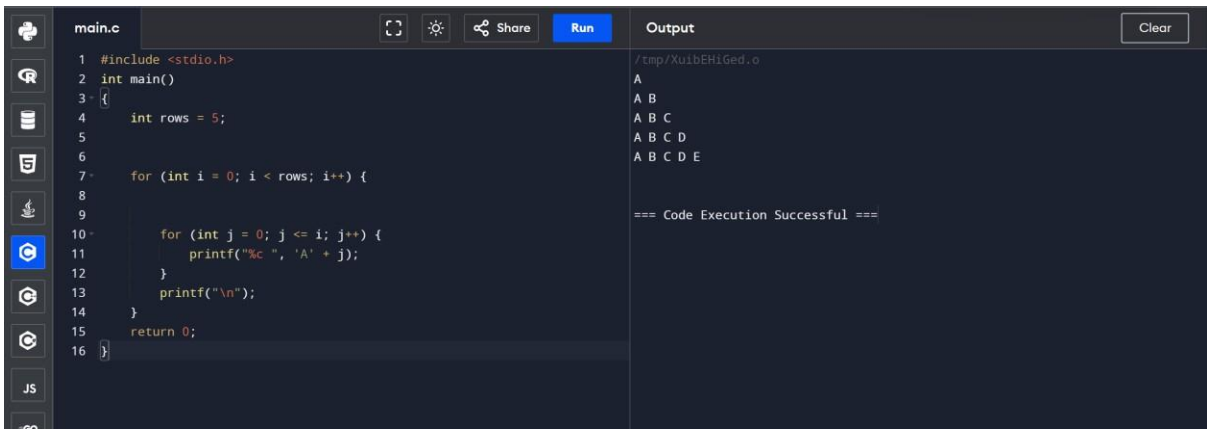
1. RIGHT HALF PYRAMID(STAR)



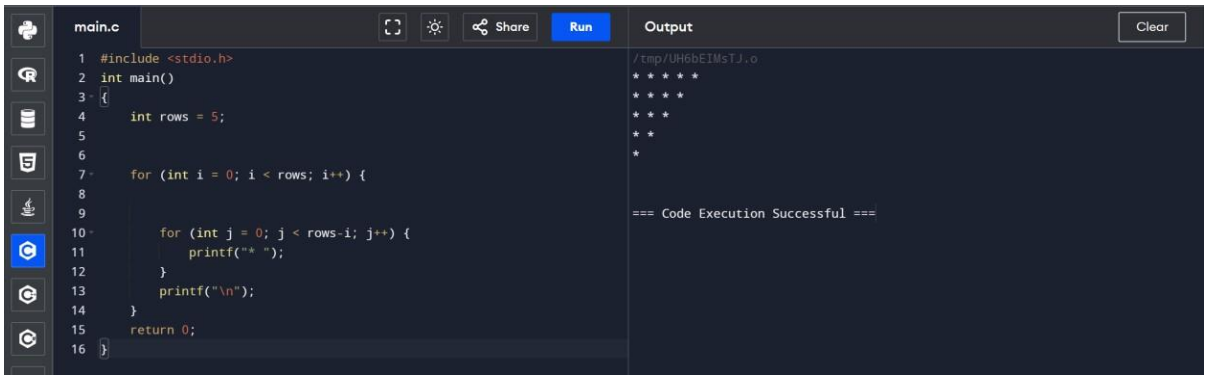
2. RIGHT HALF PYRAMID(NUMBERS)



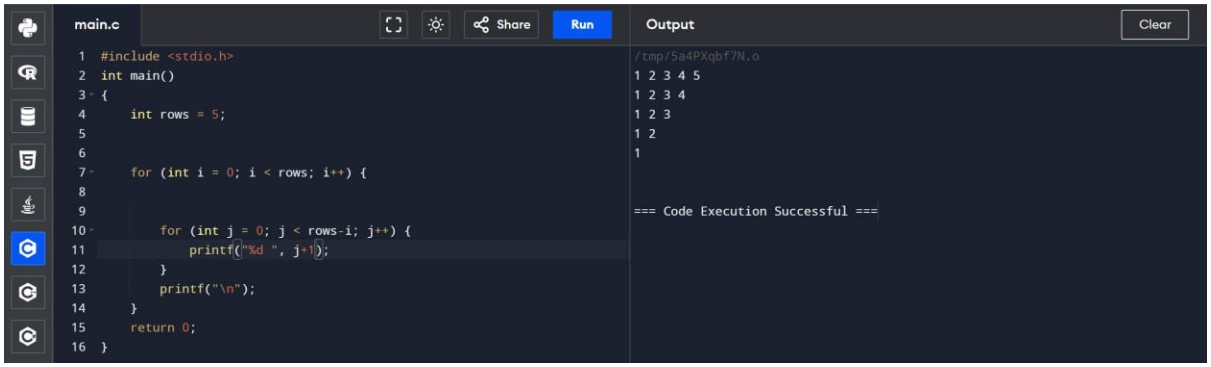
3. RIGHT HALF PYRAMID(ALPHABETS)



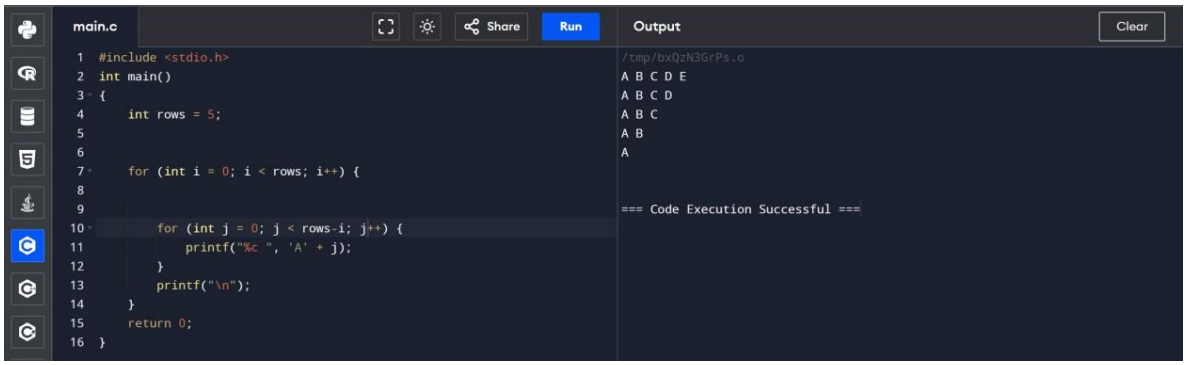
4. INVERTED RIGHT HALF PYRAMID(STAR)



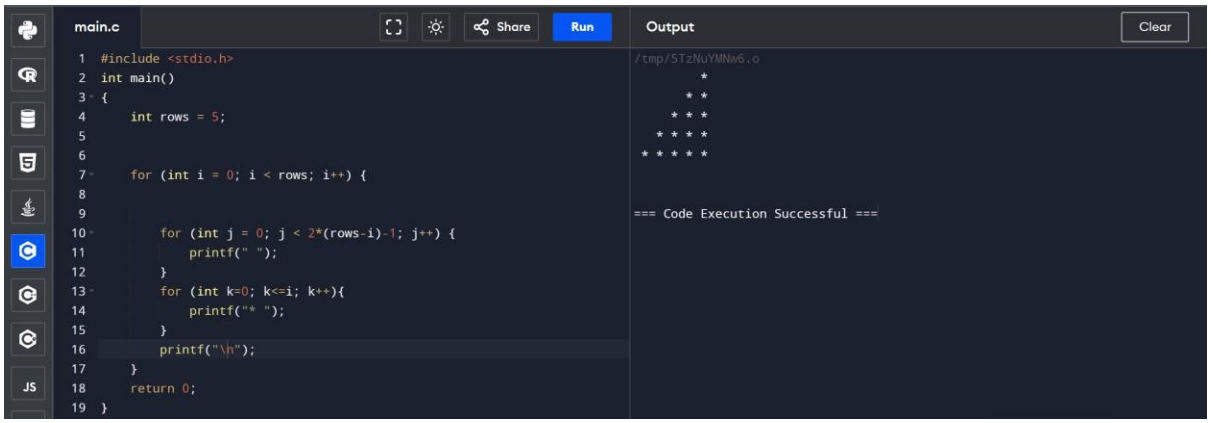
5. INVERTED RIGHT HALF PYRAMID(NUMBERS)



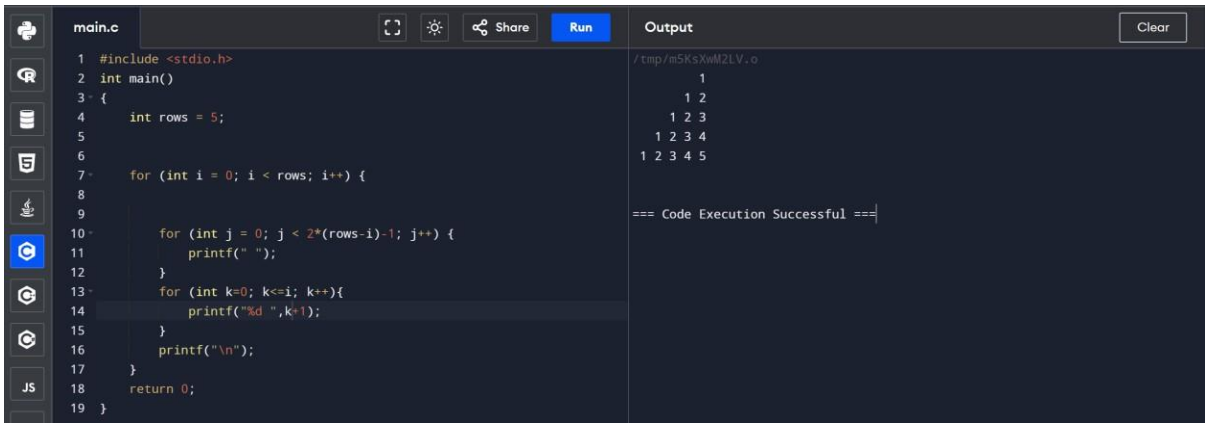
6. INVERTED RIGHT HALF PYRAMID(ALPHABETS)



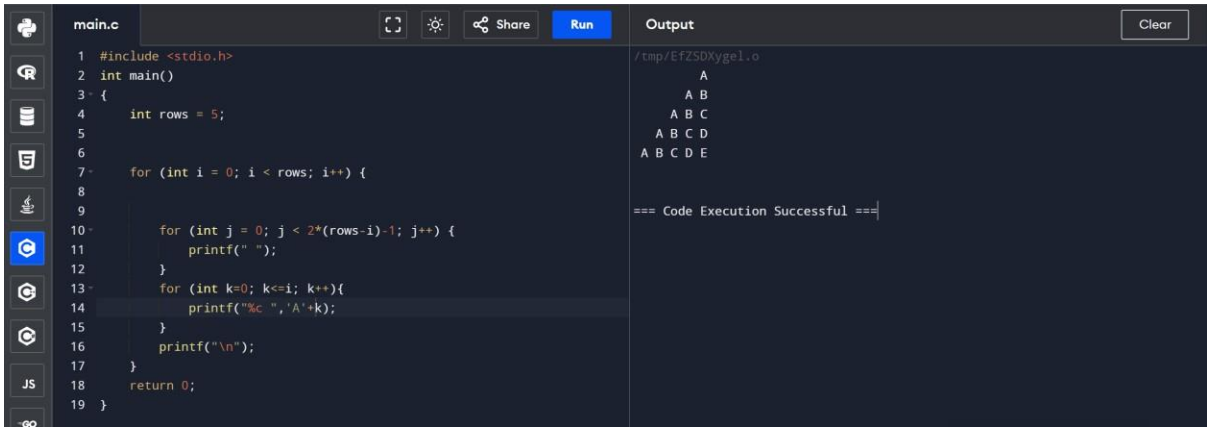
7. LEFT HALF PYRAMID(STAR)



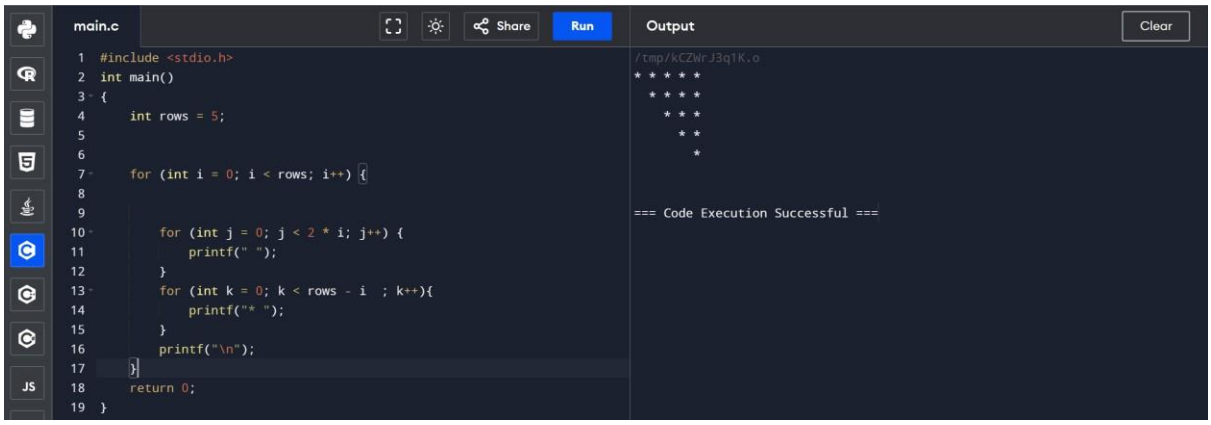
8. LEFT HALF PYRAMID(NUMBERS)



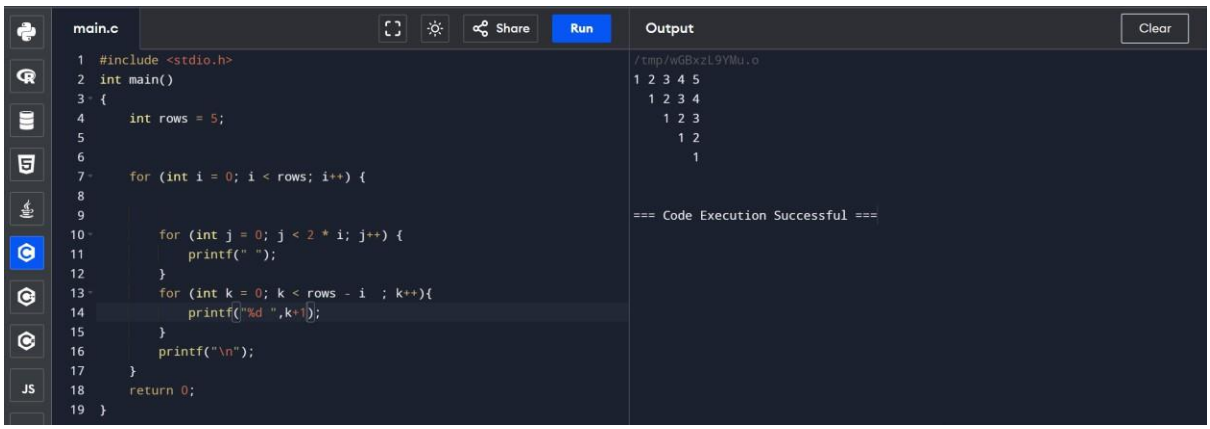
9. LEFT HALF PYRAMID(ALPHABETS)



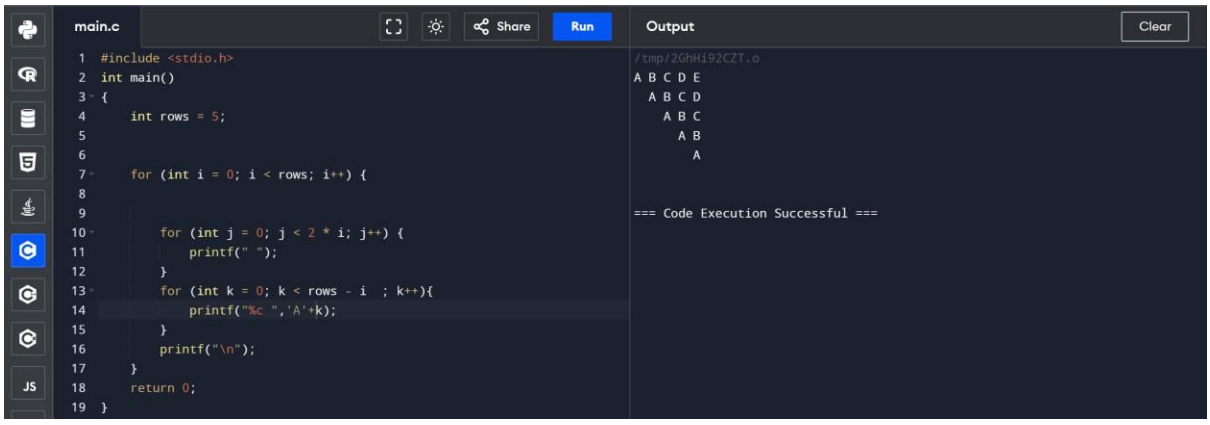
10. INVERTED LEFT HALF PYRAMID(STAR)



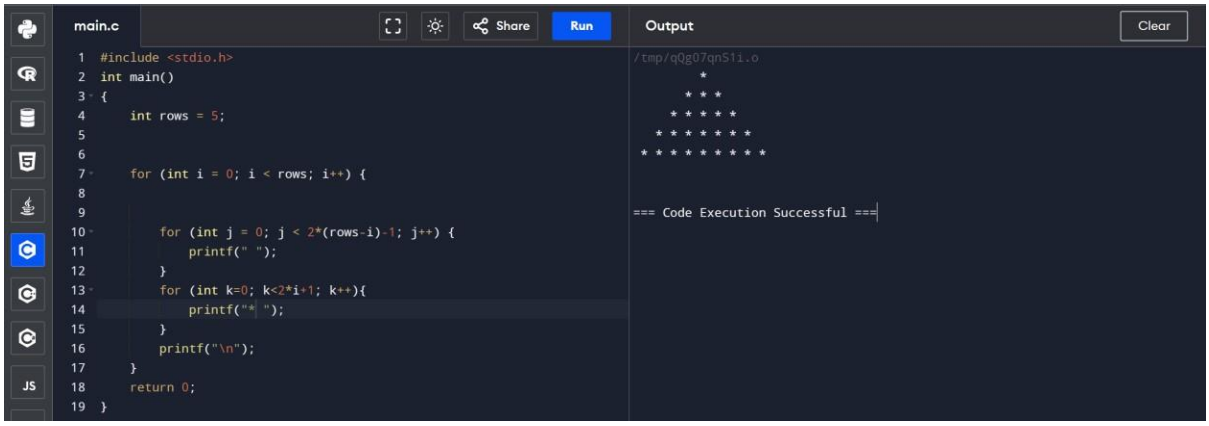
11. INVERTED LEFT HALF PYRAMID(NUMBERS)



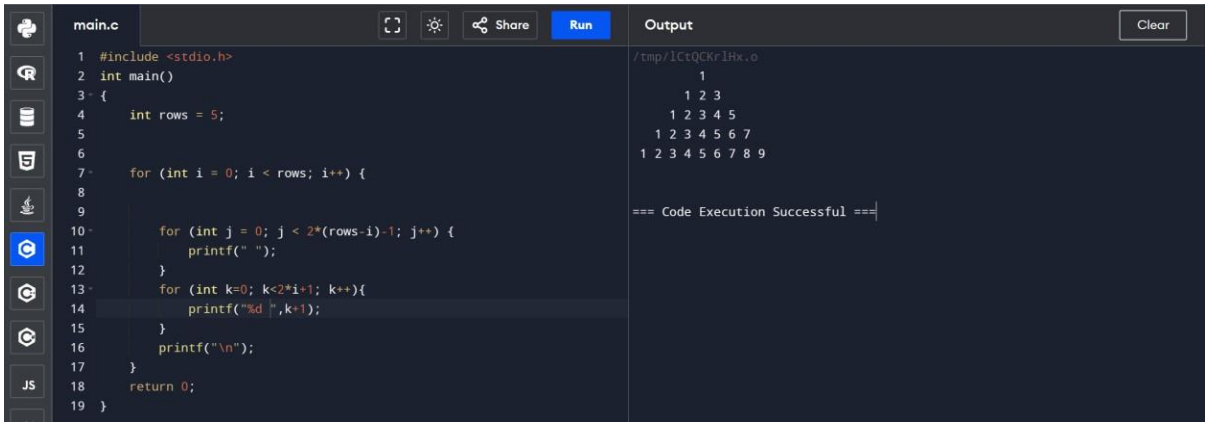
12. INVERTED LEFT HALF PYRAMID(ALPHABETS)



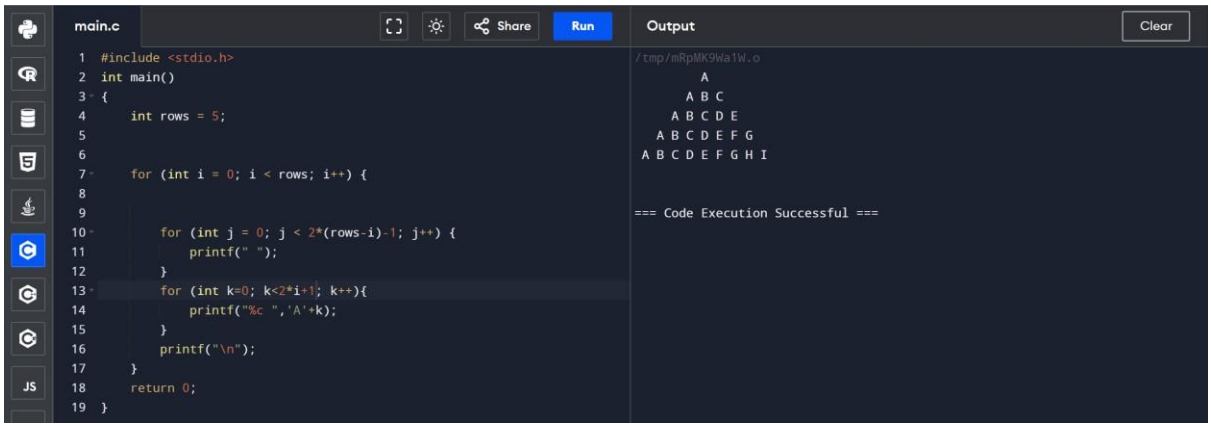
13. FULL PYRAMID(STAR)



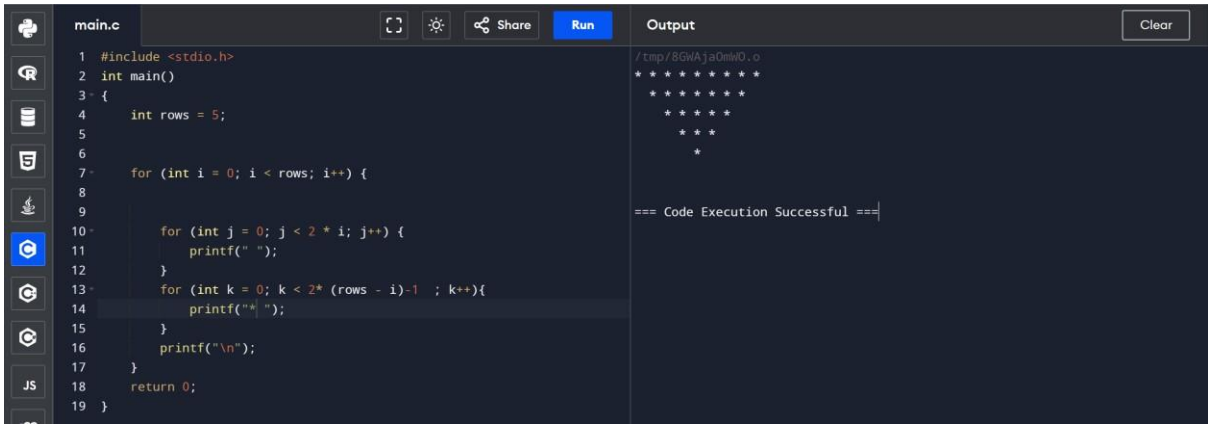
14. FULL PYRAMID(NUMBERS)



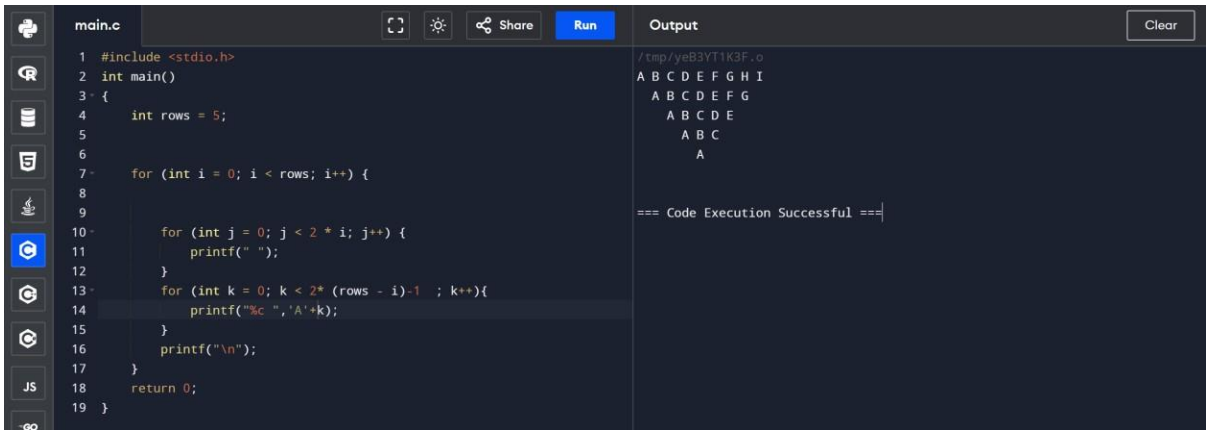
15. FULL PYRAMID(ALPHABETS)



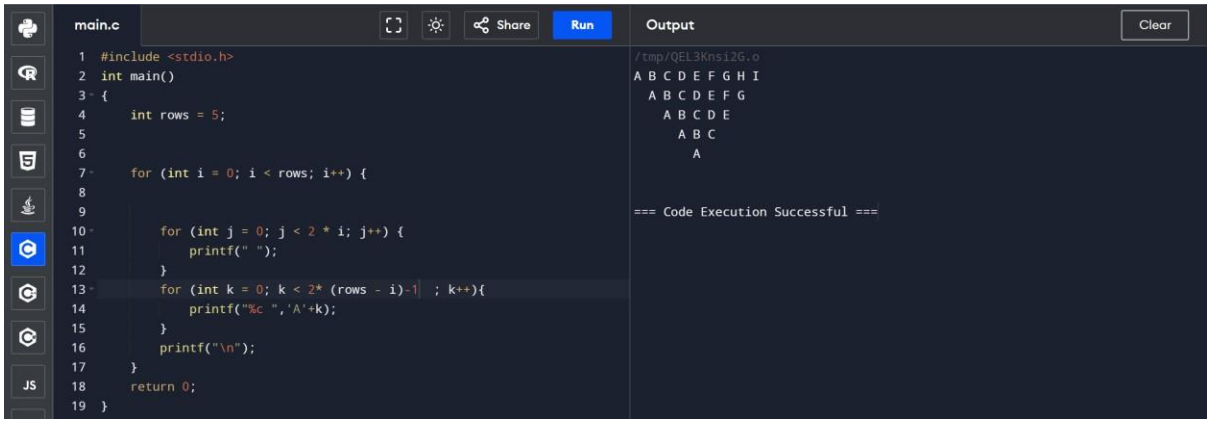
16. INVERTED PYRAMID (STAR)



17. INVERTED PYRAMID (NUMBERS)



18. INVERTED PYRAMID (ALPHABETS)



19. RHOMBUS(STAR)

main.c

```c
#include <stdio.h>
int main()
{
    int rows = 5;


    for (int i = 0; i < rows; i++) {


        for (int j = 0; j < rows-i-1; j++) {
            printf(" ");
        }
        for (int k = 0; k < rows; k++){
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

Output

```
/tmp/h4ERyqaHOD.o
    * * * * *
   * * * * *
  * * * * *
 * * * * *
* * * * *


=== Code Execution Successful ===
```

20. RHOMBUS(NUMBERS)

main.c

```c
#include <stdio.h>
int main()
{
    int rows = 5;


    for (int i = 0; i < rows; i++) {


        for (int j = 0; j < rows-i-1; j++) {
            printf(" ");
        }
        for (int k = 0; k < rows; k++){
            printf("%d ",k+1);
        }
        printf("\n");
    }
    return 0;
}
```

Output

```
/tmp/JXDVmBPZbX.o
    1 2 3 4 5
   1 2 3 4 5
  1 2 3 4 5
 1 2 3 4 5
1 2 3 4 5


=== Code Execution Successful ===
```

21. RHOMBUS(ALPHABETS)

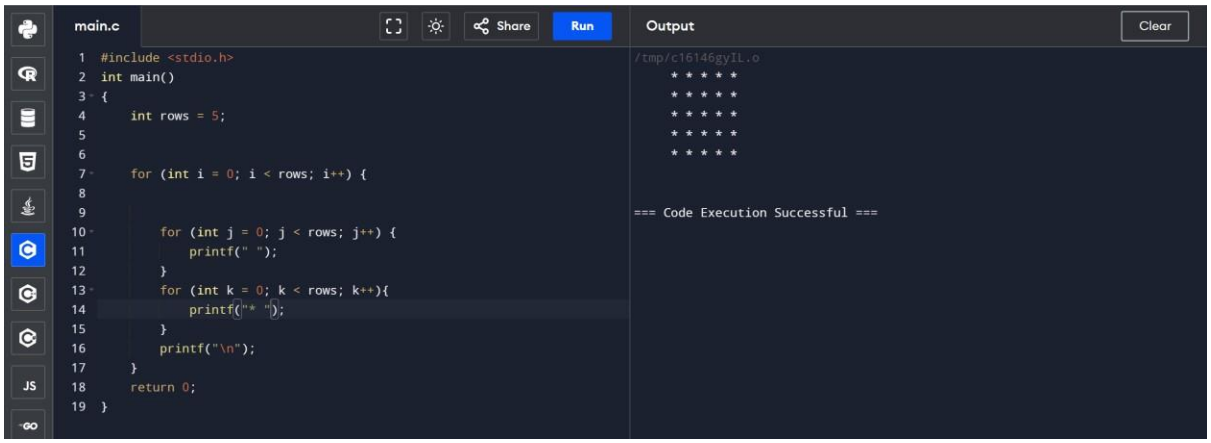
main.c

```c
#include <stdio.h>
int main()
{
    int rows = 5;


    for (int i = 0; i < rows; i++) {


        for (int j = 0; j < rows-i-1; j++) {
            printf(" ");
        }
        for (int k = 0; k < rows; k++){
            printf("%c ",'A'+k);
        }
        printf("\n");
    }
    return 0;
}
```

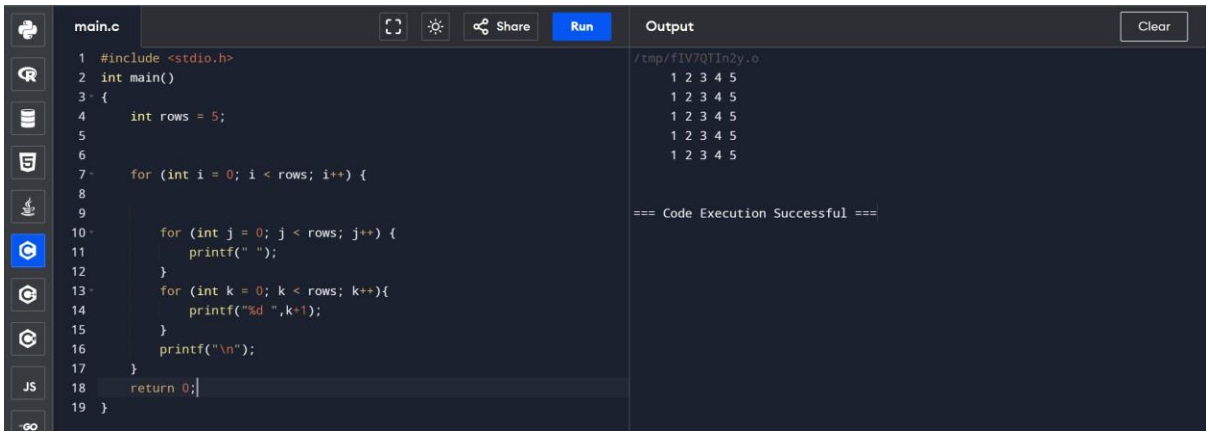
Output

```
/tmp/IlILbRsoO7.o
    A B C D E
   A B C D E
  A B C D E
 A B C D E
A B C D E


=== Code Execution Successful ===
```

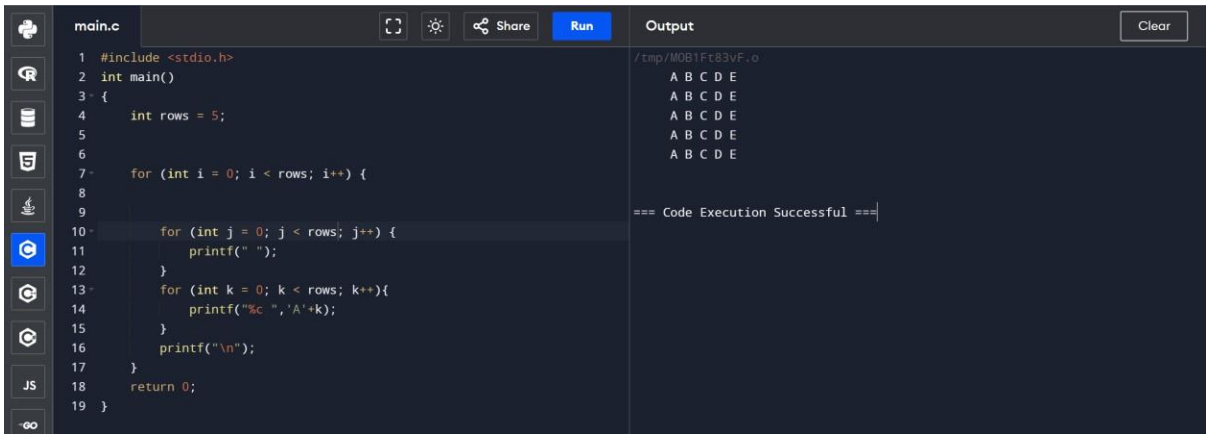
22. RECTANGLE(STARS)



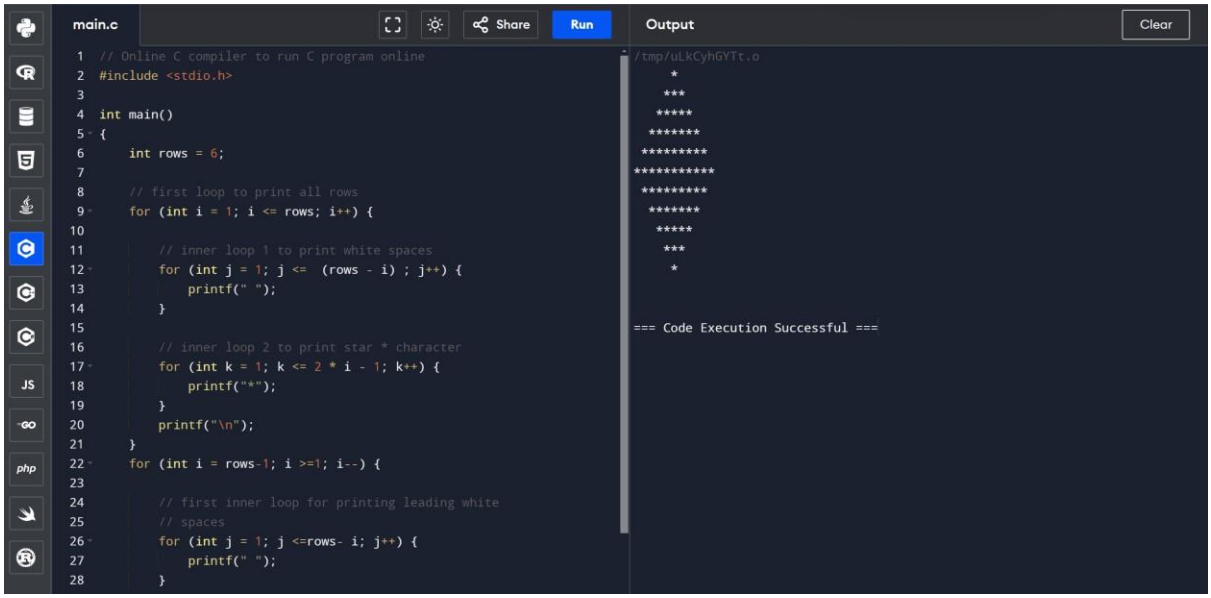
23. RECTANGLE(NUMBERS)

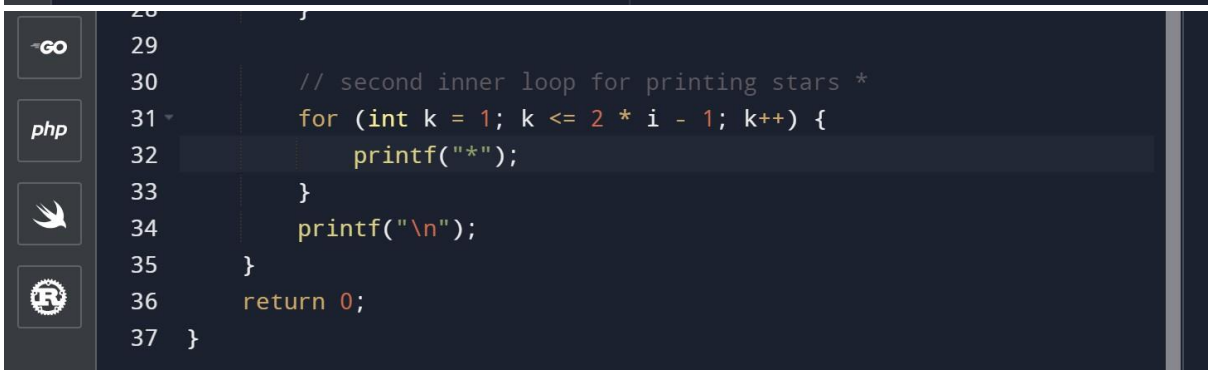


24. RECTANGLE(ALPHABETS)

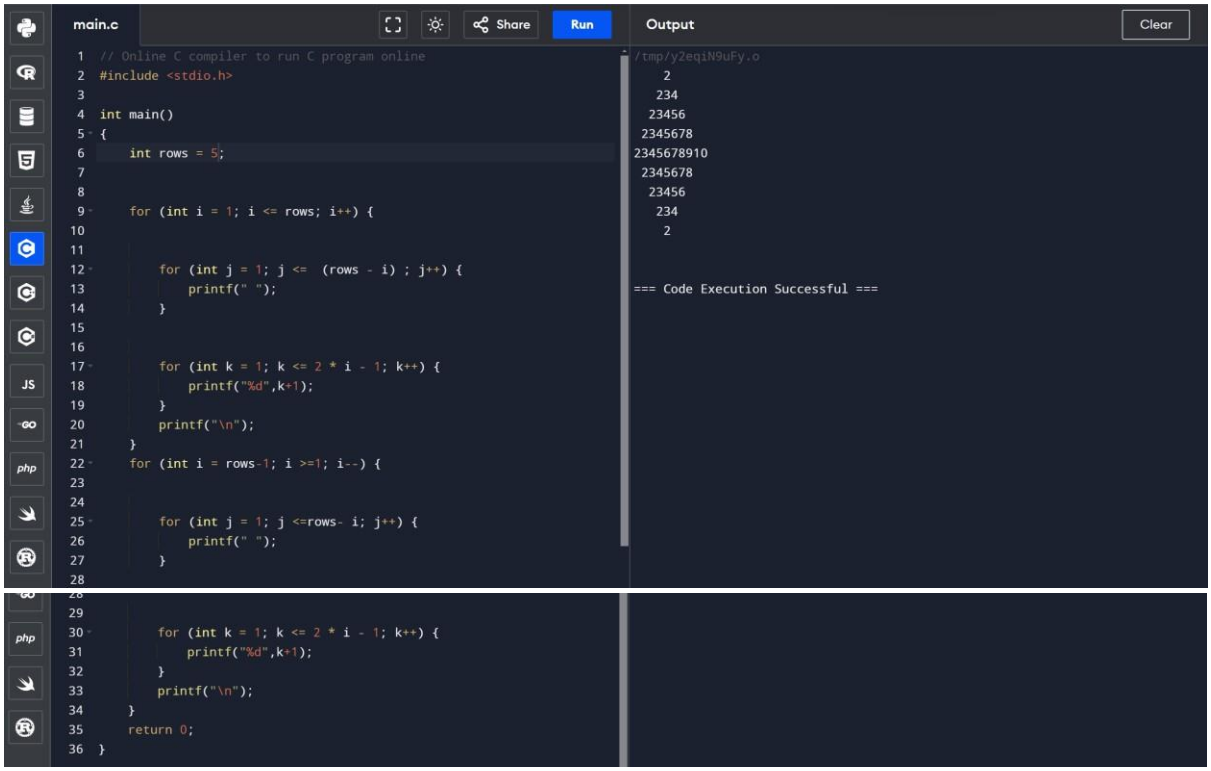


25. DIAMOND(STARS)





26. DIAMOND(NUMBERS)



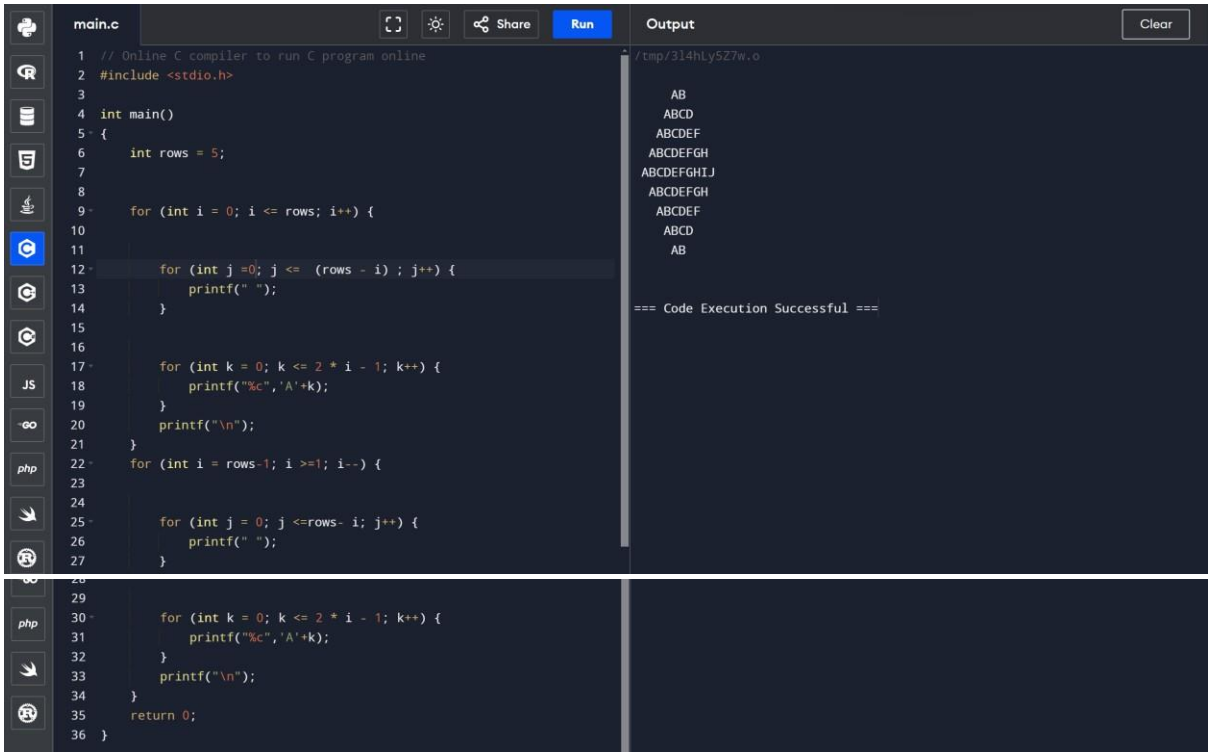
main.c

```c
// Online C compiler to run C program online
#include <stdio.h>

int main()
{
    int rows = 5;


    for (int i = 1; i <= rows; i++) {


        for (int j = 1; j <=  (rows - i) ; j++) {
            printf(" ");
        }


        for (int k = 1; k <= 2 * i - 1; k++) {
            printf("%d",k+1);
        }
        printf("\n");
    }
    for (int i = rows-1; i >=1; i--) {


        for (int j = 1; j <=rows- i; j++) {
            printf(" ");
        }


        for (int k = 1; k <= 2 * i - 1; k++) {
            printf("%d",k+1);
        }
        printf("\n");
    }
    return 0;
}
```

Output

```
/tmp/y2eqiN9uFy.o
    2
   234
  23456
 2345678
2345678910
 2345678
  23456
   234
    2


=== Code Execution Successful ===
```

27. DIAMOND(ALPHABETS)



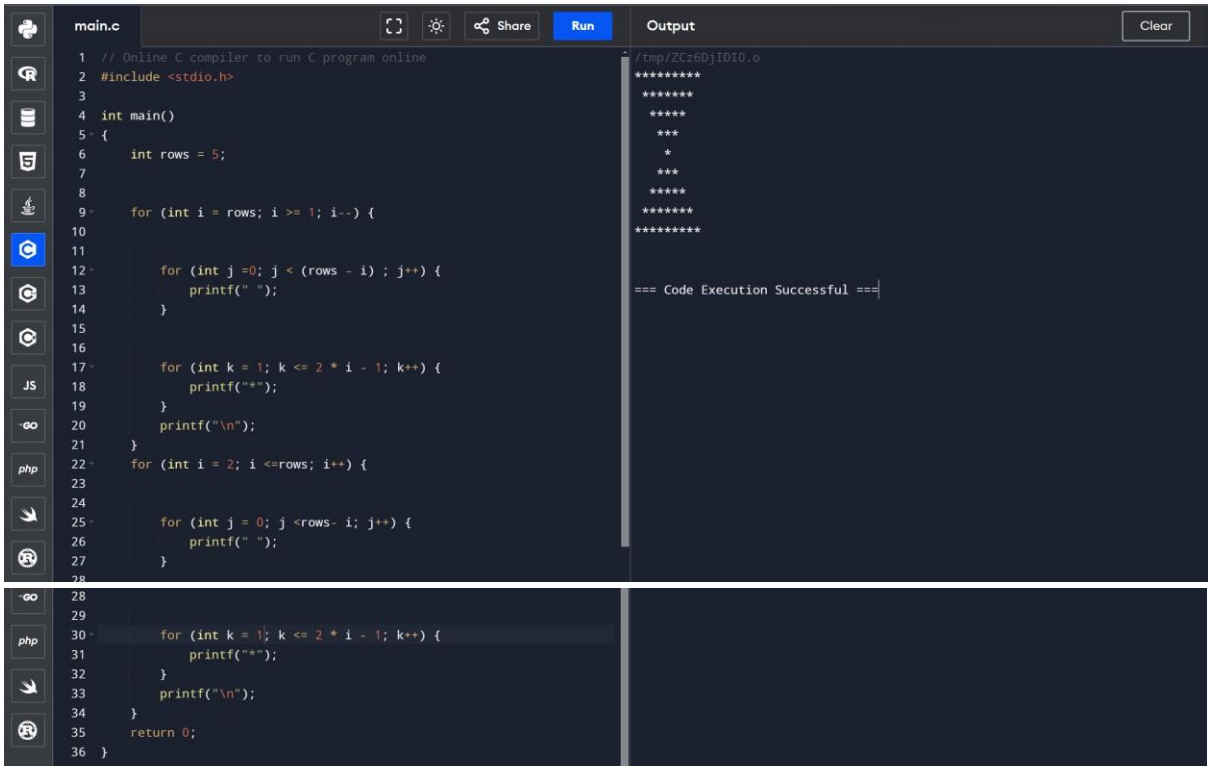
main.c

```c
// Online C compiler to run C program online
#include <stdio.h>

int main()
{
    int rows = 5;


    for (int i = 0; i <= rows; i++) {


        for (int j =0; j <=  (rows - i) ; j++) {
            printf(" ");
        }


        for (int k = 0; k <= 2 * i - 1; k++) {
            printf("%c",'A'+k);
        }
        printf("\n");
    }
    for (int i = rows-1; i >=1; i--) {


        for (int j = 0; j <=rows- i; j++) {
            printf(" ");
        }


        for (int k = 0; k <= 2 * i - 1; k++) {
            printf("%c",'A'+k);
        }
        printf("\n");
    }
    return 0;
}
```

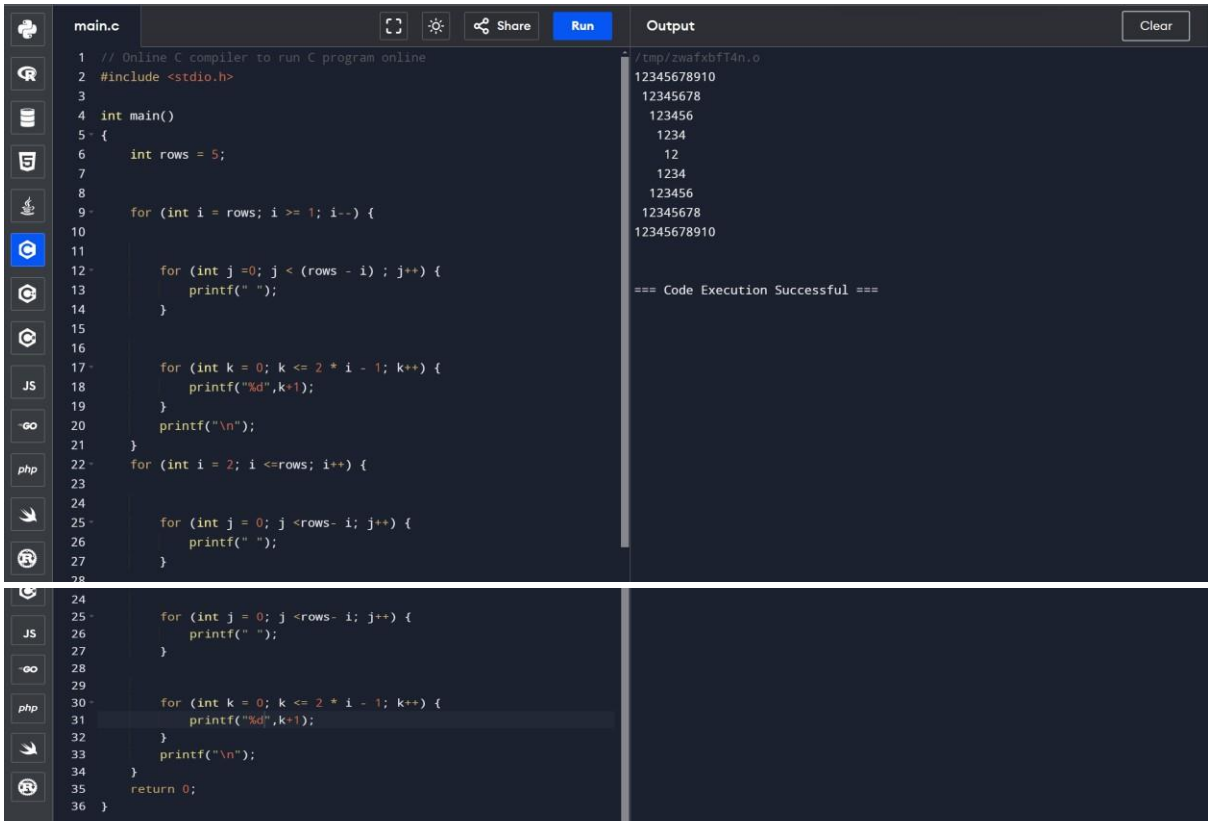
Output

```
/tmp/314hLySZ7w.o

    AB
   ABCD
  ABCDEF
 ABCDEFGH
ABCDEFGHIJ
 ABCDEFGH
  ABCDEF
   ABCD
    AB


=== Code Execution Successful ===
```

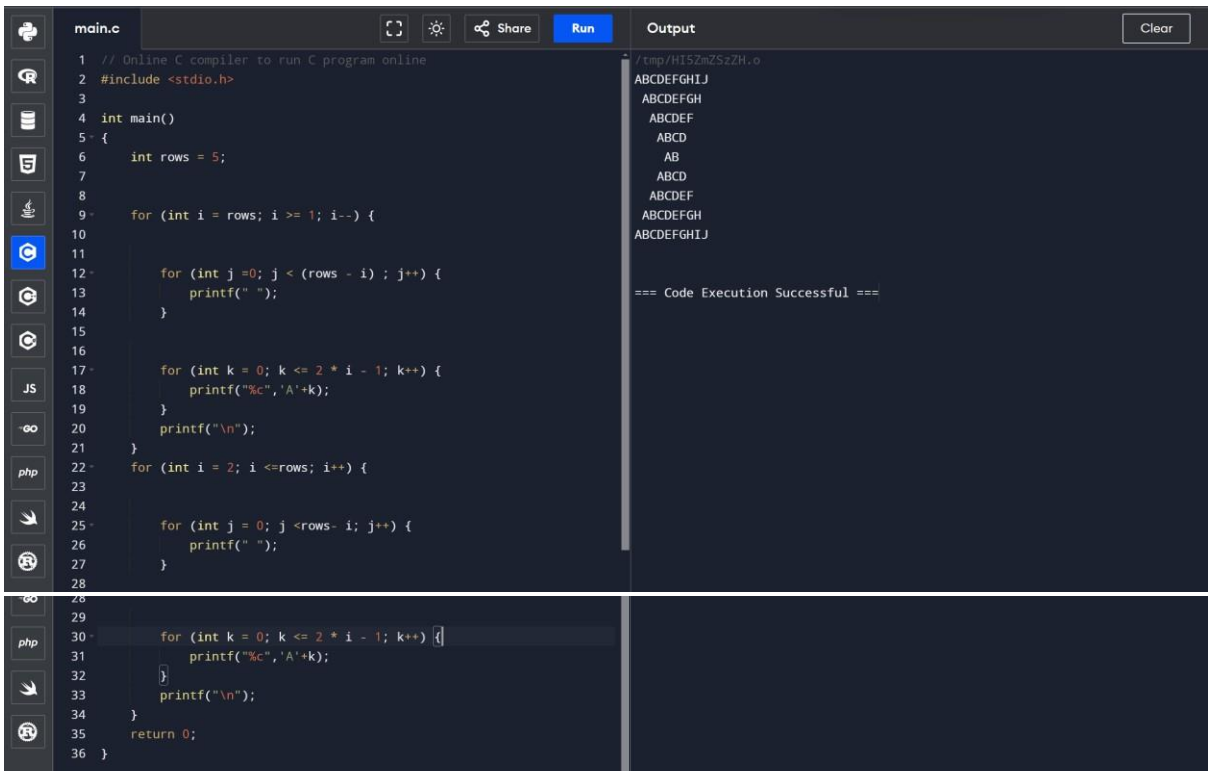
28. HOURGLASS(STAR)



29. HOURGLASS(NUMBERS)



30. HOURGLASS(ALPHABETS)



31. PASCAL'S TRIANGLE

main.c

Share Run Output

```c
int main() {
    int rows, coef = 1, i, j;
    printf("Enter number of rows: ");
    scanf("%d", &rows);

    for(i = 0; i < rows; i++) {
        for(j = 1; j <= rows - i; j++) {
            printf(" ");
        }
        for(j = 0; j <= i; j++) {
            if (j == 0 || i == j) {
                coef = 1;
            } else {
                coef = coef * (i - j + 1) / j;
            }
            printf("%4d", coef);
        }
        printf("\n");
    }

    return 0;
}
```

```
/tmp/Eoqx4tMCMz.o
Enter number of rows: 5
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1


=== Code Execution Successful ===
```

32. HOLLOW DIAMOND(STARS)

main.c

Share Run

```c
#include <stdio.h>

int main() {
    int i, j, rows;
    printf("Enter number of rows: ");
    scanf("%d", &rows);

    // Upper half
    for(i = 1; i <= rows; i++) {
        for(j = 1; j <= rows - i; j++) {
            printf(" ");
        }
        for(j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1) {
                printf("*");
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    // Lower half
    for(i = rows - 1; i >= 1; i--) {
        for(j = 1; j <= rows - i; j++) {
            printf(" ");
        }
        for(j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1) {
                printf("*");
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

Output

```
/tmp/wj6HqjEUPj.o
Enter number of rows: 7
      *
     * *
    *   *
   *     *
  *       *
 *         *
*           *
 *         *
  *       *
   *     *
    *   *
     * *
      *


=== Code Execution Successful ===
```

33. HOLLOW DIAMOND(NUMBERS)

main.c

Share Run

```c
#include <stdio.h>

int main() {
    int i, j, n;
    printf("Enter the number of rows: ");
    scanf("%d", &n);

    // Upper half
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1) {
                printf("%d", i);
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    // Lower half
    for (i = n - 1; i >= 1; i--) {
        for (j = 1; j <= n - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1) {
                printf("%d", i);
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

Output

```
/tmp/hjJHqoeOjv.o
Enter the number of rows: 10
         1
        2 2
       3   3
      4     4
     5       5
    6         6
   7           7
  8             8
 9               9
10                 10
 9               9
  8             8
   7           7
    6         6
     5       5
      4     4
       3   3
        2 2
         1


=== Code Execution Successful ===
```

```c
    for (j = 1; j <= n - i; j++) {
        printf(" ");
    }
    for (j = 1; j <= 2 * i - 1; j++) {
        if (j == 1 || j == 2 * i - 1) {
            printf("%d", i);
        } else {
            printf(" ");
        }
    }
    printf("\n");
}

return 0;
}
```

```
     5       5
    6         6
   7           7
  8             8
 9               9
10                 10
 9               9
  8             8
   7           7
    6         6
     5       5
      4     4
       3   3
        2 2
         1


=== Code Execution Successful ===
```

34. HOLLOW DIAMOND(ALPHABETS)

main.c

Share Run

```c
#include <stdio.h>

int main() {
    int i, j, n;
    char ch;
    printf("Enter the number of rows: ");
    scanf("%d", &n);

    // Upper half
    for (i = 1; i <= n; i++) {
        ch = 'A' + i - 1;
        for (j = 1; j <= n - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1) {
                printf("%c", ch);
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    // Lower half
    for (i = n - 1; i >= 1; i--) {
        ch = 'A' + i - 1;
        for (j = 1; j <= n - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1) {
                printf("%c", ch);
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

Output

```
/tmp/bjvcuotbRr.o
Enter the number of rows: 5
    A
   B B
  C   C
 D     D
E       E
 D     D
  C   C
   B B
    A


=== Code Execution Successful ===
```

```
Enter the number of rows: 5
    A
   B B
  C   C
 D     D
E       E
 D     D
  C   C
   B B
    A


=== Code Execution Successful ===
```

35. HOLLOW PYRAMID(STARS)

main.c | Share | Run | Output

```c
#include <stdio.h>

int main() {
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; i++) {
        for (j = 1; j <= rows - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1 || i == rows) {
                printf("*");
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

```
/tmp/CJzFpmZLiV.o
Enter the number of rows: 6
     *
    * *
   *   *
  *     *
 *       *
***********


=== Code Execution Successful ===
```

36. HOLLOW PYRAMID(NUMBERS)

main.c | Share | Run | Output

```c
#include <stdio.h>

int main() {
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; i++) {
        for (j = 1; j <= rows - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1 || i == rows) {
                printf("%d", i);
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

```
/tmp/MOOire2wwN.o
Enter the number of rows: 5
    1
   2 2
  3   3
 4     4
555555555


=== Code Execution Successful ===
```

37. HOLLOW PYRAMID(ALPHABET)

main.c — Share — Run — Output

```c
#include <stdio.h>

int main() {
    int i, j, rows;
    char ch;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; i++) {
        for (j = 1; j <= rows - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1 || i == rows) {
                printf("%c", 'A' + i - 1);
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

```
/tmp/Vd9ydCaKu8.o
Enter the number of rows: 8
       A
      B B
     C   C
    D     D
   E       E
  F         F
 G           G
HHHHHHHHHHHHHHH


=== Code Execution Successful ===
```

38. HOLLOW FULL PYRAMID(STARS)

main.c — Share — Run — Output

```c
#include <stdio.h>

int main() {
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; i++) {
        for (j = 1; j <= rows - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1 || i == rows) {
                printf("*");
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

```
/tmp/Ntfdw6azvR.o
Enter the number of rows: 5
    *
   * *
  *   *
 *     *
*********


=== Code Execution Successful ===
```

39. HOLLOW FULL PYRAMID(NUMBERS)

main.c

```c
#include <stdio.h>

int main() {
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; i++) {
        for (j = 1; j <= rows - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1 || i == rows) {
                printf("%d", i);
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

Output

```
/tmp/rvtj2Hq7EV.o
Enter the number of rows: 7
      1
     2 2
    3   3
   4     4
  5       5
 6         6
7777777777777


=== Code Execution Successful ===
```

40. HOLLOW FULL PYRAMID(ALPHABETS)

main.c

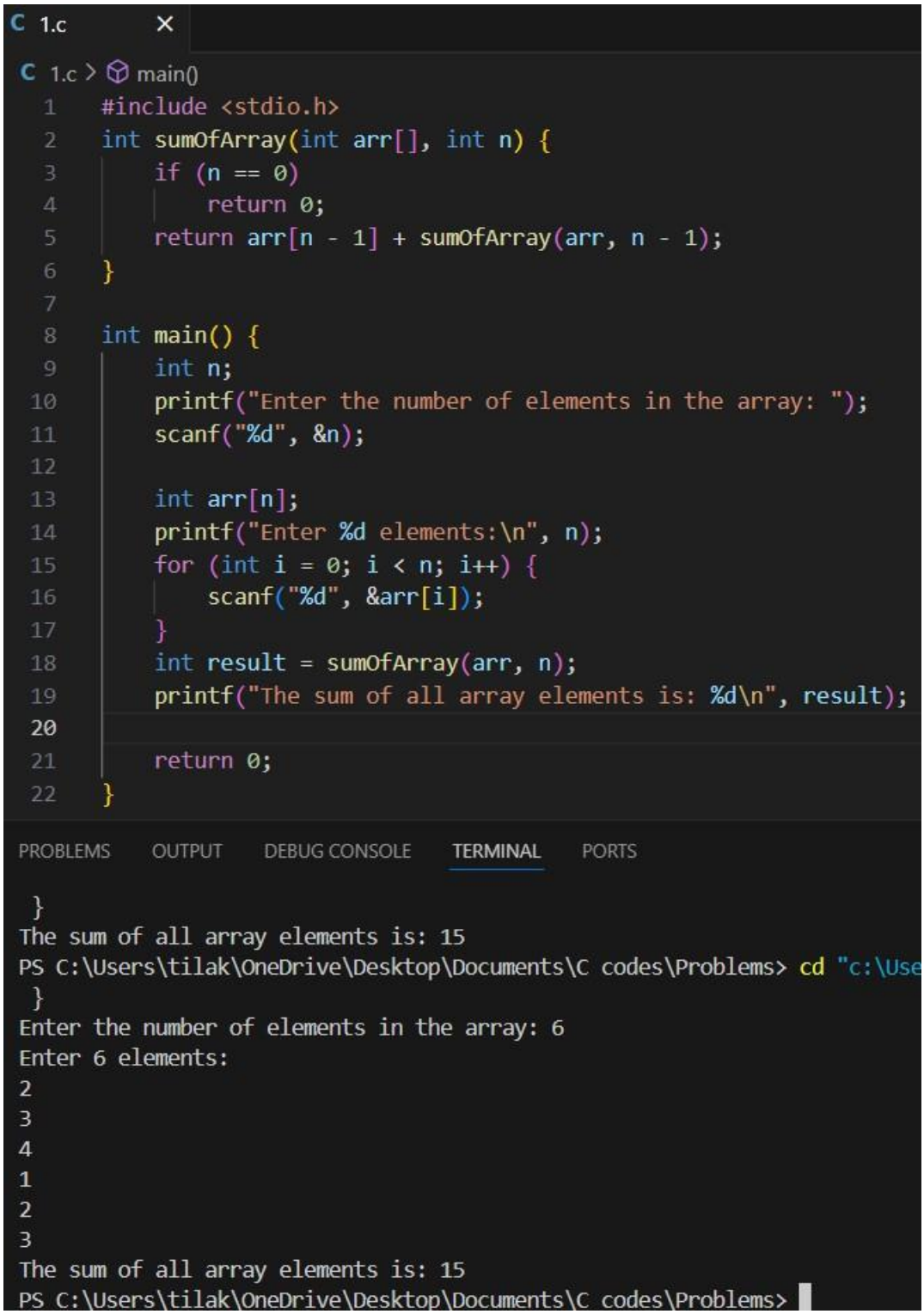
```c
#include <stdio.h>

int main() {
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; i++) {
        for (j = 1; j <= rows - i; j++) {
            printf(" ");
        }
        for (j = 1; j <= 2 * i - 1; j++) {
            if (j == 1 || j == 2 * i - 1 || i == rows) {
                printf("%c", 'A' + i - 1);
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

Output

```
/tmp/ITaeJHwxQc.o
Enter the number of rows: 9
        A
       B B
      C   C
     D     D
    E       E
   F         F
  G           G
 H             H
IIIIIIIIIIIIIIIII


=== Code Execution Successful ===
```

- LAB-1

1. Find sum of all array elements using recursion.



2. Create an array ‘a1’ with ‘n’ elements. Insert an element in ith position of ‘a1’ a nd also delete an element from jth position of ‘a1’.

C 2.c

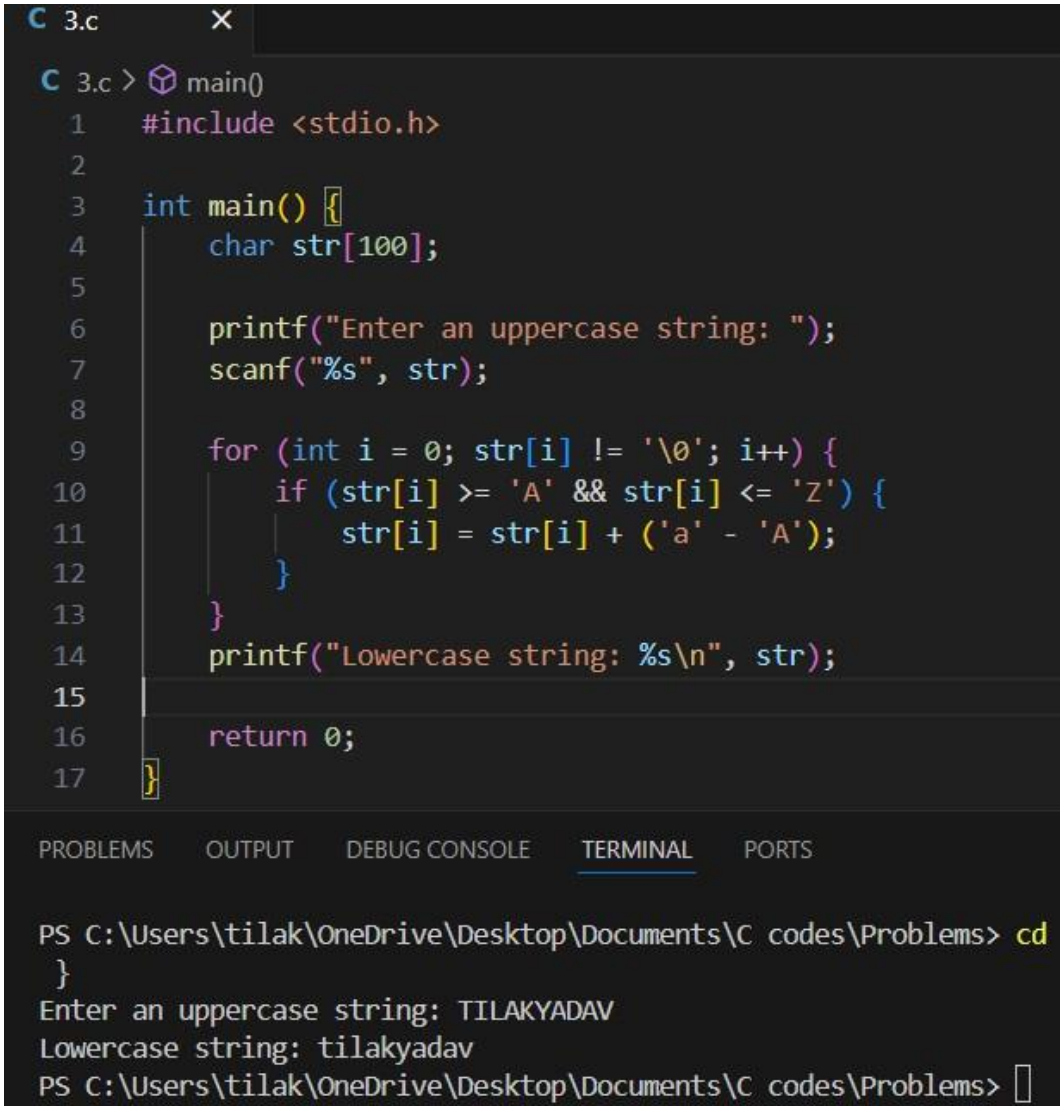
C 2.c > ...

```c
#include <stdio.h>
int main() {
    int n, i, j, element;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[100];

    printf("Enter %d elements:\n", n);
    for (int k = 0; k < n; k++) {
        scanf("%d", &arr[k]);
    }

    printf("Enter the position (0-based index) to insert an element: ");
    scanf("%d", &i);
    printf("Enter the element to insert: ");
    scanf("%d", &element);

    for (int k = n; k > i; k--) {
        arr[k] = arr[k - 1];
    }
    arr[i] = element;
    n++;

    printf("Array after insertion:\n");
    for (int k = 0; k < n; k++) {
        printf("%d ", arr[k]);
    }
    printf("\n");

    printf("Enter the position (0-based index) to delete an element: ");
    scanf("%d", &j);

    for (int k = j; k < n - 1; k++) {
        arr[k] = arr[k + 1];
    }
    n--;

    printf("Array after deletion:\n");
    for (int k = 0; k < n; k++) {
        printf("%d ", arr[k]);
```

```c
    printf("Array after deletion:\n");
    for (int k = 0; k < n; k++) {
        printf("%d ", arr[k]);
    }
    printf("\n");

    return 0;
}

```

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\U
 }
Enter the number of elements in the array: 5
Enter 5 elements:
1
2
3
4
5
Enter the position (0-based index) to insert an element: 3
Enter the element to insert: 9
Array after insertion:
1 2 3 9 4 5
Enter the position (0-based index) to delete an element: 2
Array after deletion:
1 2 9 4 5
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

3. Convert uppercase string to lowercase using for loop.



4. Find the sum of rows and columns of matrix of given order (row x column).

C 4.c ●

C 4.c > ...

```c
#include <stdio.h>
int main() {
    int rows, columns;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &columns);

    int matrix[10][10];
    printf("Enter the elements of the matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    printf("Sum of each row:\n");
    for (int i = 0; i < rows; i++) {
        int rowSum = 0;
        for (int j = 0; j < columns; j++) {
            rowSum += matrix[i][j];
        }
        printf("Row %d: %d\n", i + 1, rowSum);
    }

    printf("Sum of each column:\n");
    for (int j = 0; j < columns; j++) {
        int colSum = 0;
        for (int i = 0; i < rows; i++) {
            colSum += matrix[i][j];
        }
        printf("Column %d: %d\n", j + 1, colSum);
    }

    return 0;
}

```

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\
 }
Enter the number of rows: 3
Enter the number of columns: 2
Enter the elements of the matrix:
2
3
5
7
8
3
Sum of each row:
Row 1: 5
Row 2: 12
Row 3: 11
Sum of each column:
Column 1: 15
Column 2: 13
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

5. Find the product of two matrices using pointers.

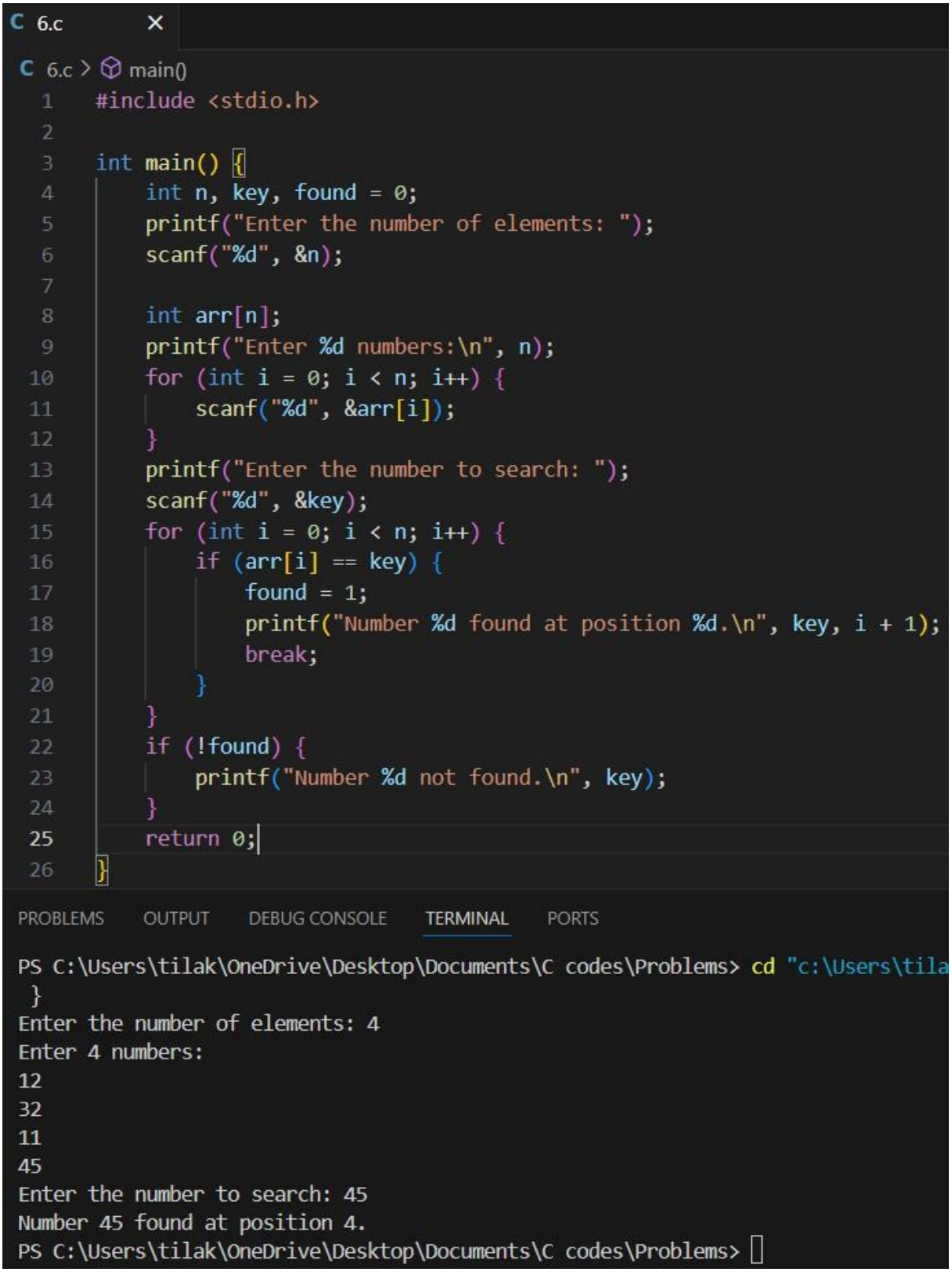
5.c

```c
#include <stdio.h>

void multiplyMatrices(int *mat1, int *mat2, int *result, int r1, int c1, int c2) {
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            *(result + i * c2 + j) = 0;
            for (int k = 0; k < c1; k++) {
                *(result + i * c2 + j) += *(mat1 + i * c1 + k) * *(mat2 + k * c2 + j);
            }
        }
    }
}
int main() {
    int r1, c1, r2, c2;
    printf("Enter rows and columns of first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter rows and columns of second matrix: ");
    scanf("%d %d", &r2, &c2);

    if (c1 != r2) {
        printf("Matrix multiplication not possible. Columns of first matrix must match rows of second matrix.\n");
        return 1;
    }

    int mat1[r1][c1], mat2[r2][c2], result[r1][c2];
    printf("Enter elements of first matrix (%dx%d):\n", r1, c1);
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c1; j++) {
            scanf("%d", &mat1[i][j]);
        }
    }
    printf("Enter elements of second matrix (%dx%d):\n", r2, c2);
    for (int i = 0; i < r2; i++) {
        for (int j = 0; j < c2; j++) {
            scanf("%d", &mat2[i][j]);
        }
    }
    multiplyMatrices(&mat1[0][0], &mat2[0][0], &result[0][0], r1, c1, c2);
    printf("Product of the matrices:\n");
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            printf("%d ", result[i][j]);
```

```c
    multiplyMatrices(&mat1[0][0], &mat2[0][0], &result[0][0], r1, c1, c2);
    printf("Product of the matrices:\n");
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c
 }
Enter rows and columns of first matrix: 2
3
Enter rows and columns of second matrix: 3
2
Enter elements of first matrix (2x3):
2
4
6
4
7
8
Enter elements of second matrix (3x2):
6
4
12
9
6
5
Product of the matrices:
96 74
156 119
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

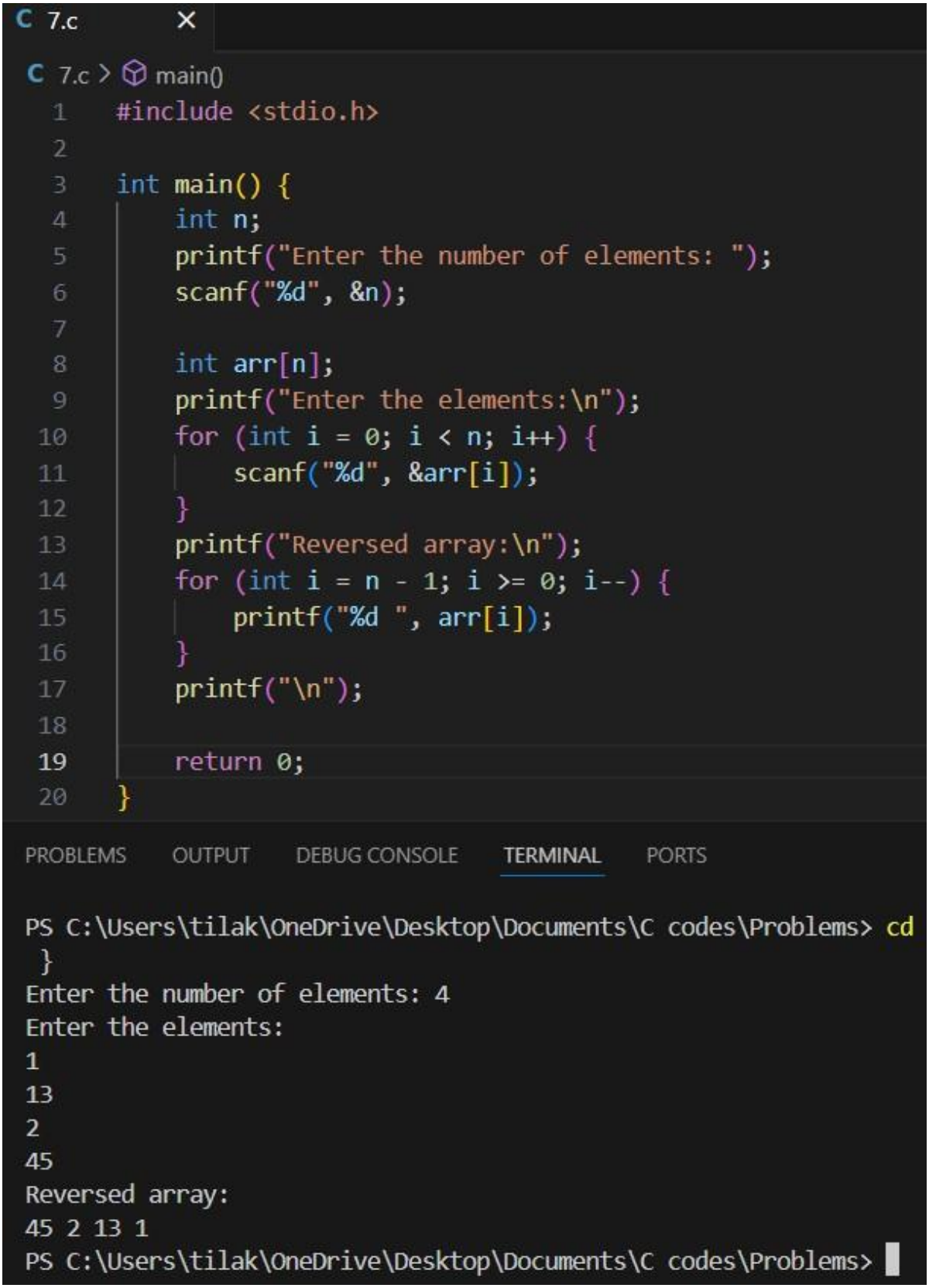
6. Store ‘n’ numbers (integers or real) in an array. Conduct a linear search for a given number and report success or failure in the form of a suitable message.



```c
#include <stdio.h>

int main() {
    int n, key, found = 0;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d numbers:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the number to search: ");
    scanf("%d", &key);
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            found = 1;
            printf("Number %d found at position %d.\n", key, i + 1);
            break;
        }
    }
    if (!found) {
        printf("Number %d not found.\n", key);
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\Users\tila
 }
Enter the number of elements: 4
Enter 4 numbers:
12
32
11
45
Enter the number to search: 45
Number 45 found at position 4.
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

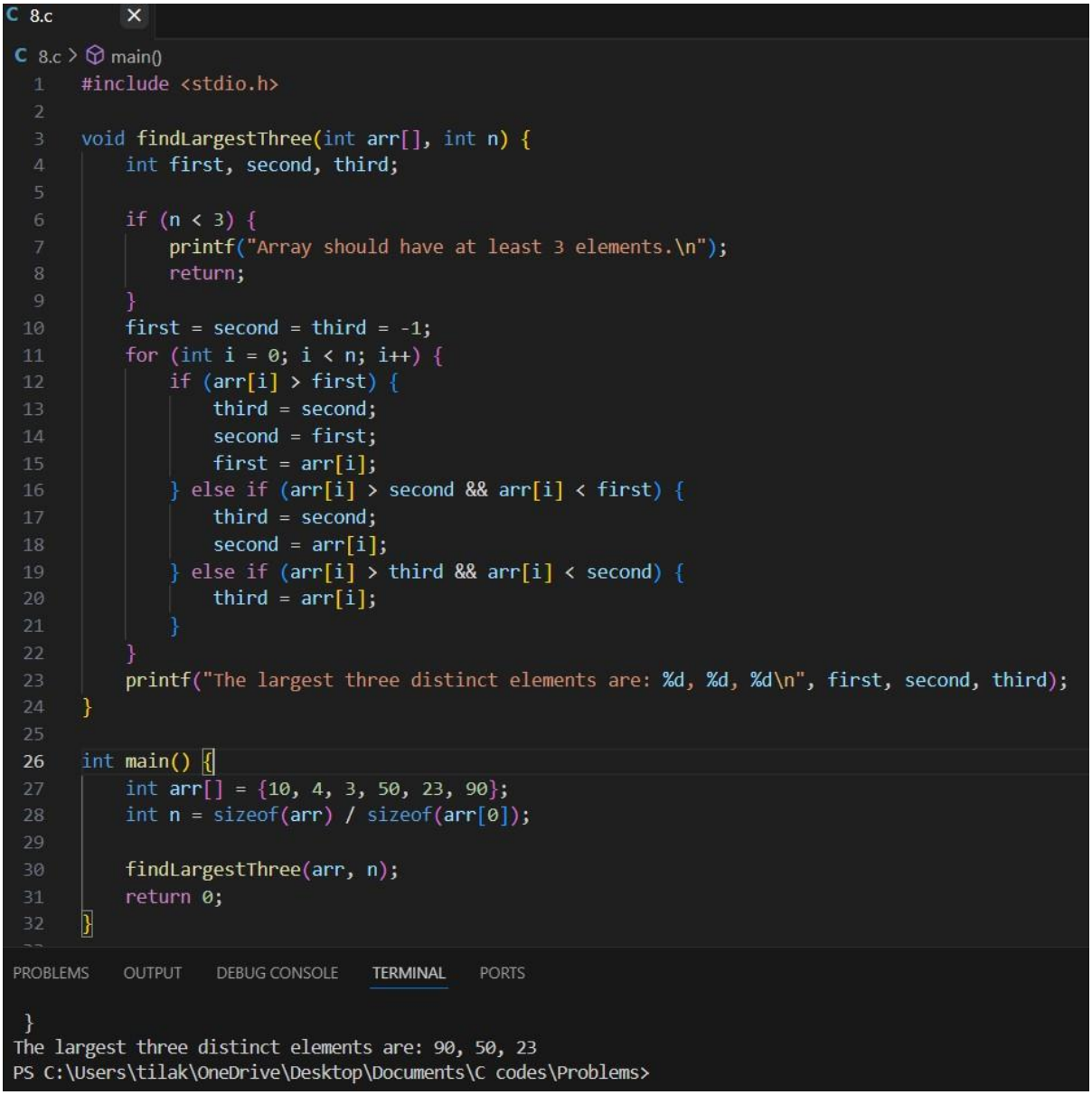
7. Write a program to reverse an array.



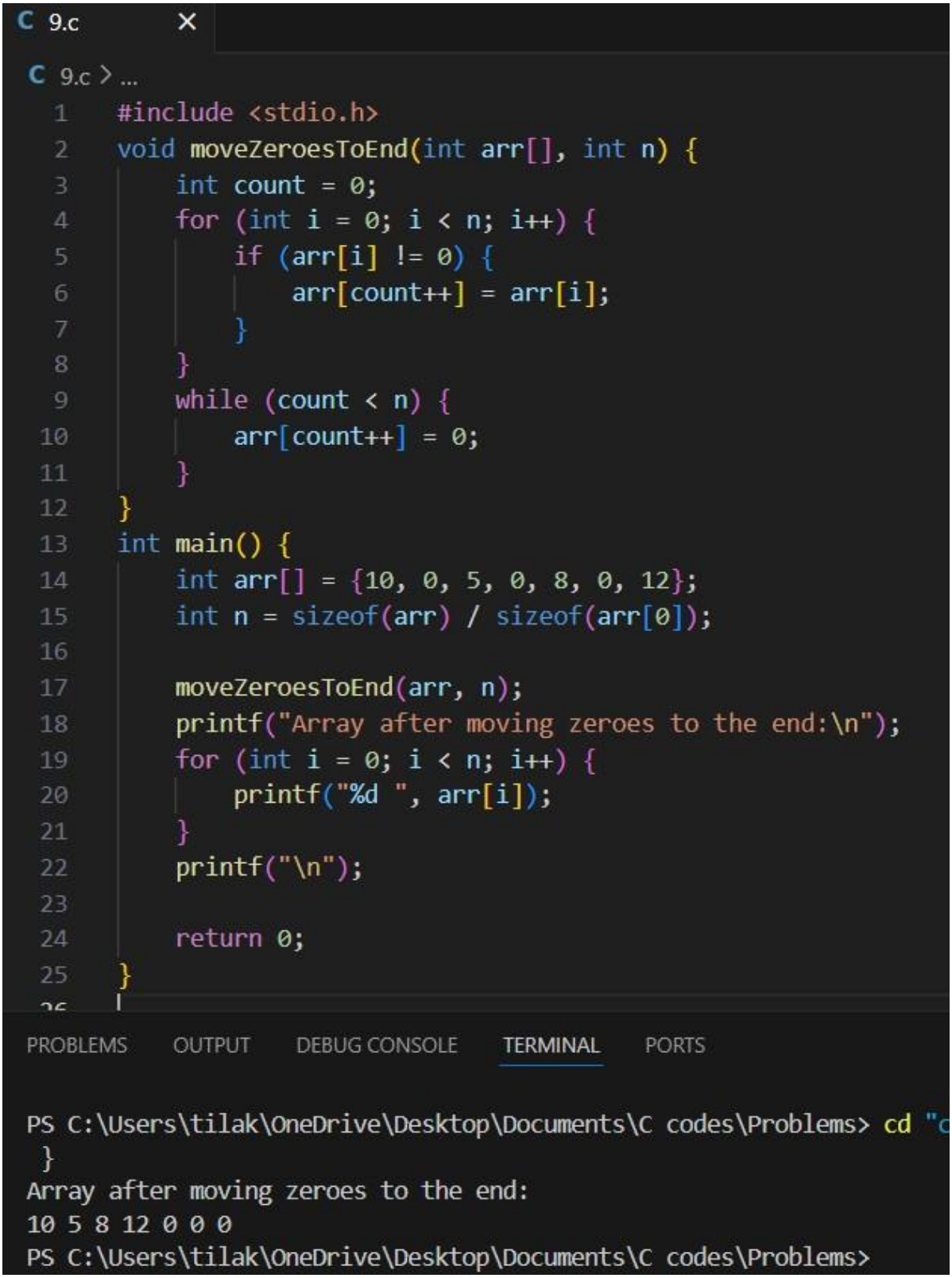
```c
#include <stdio.h>

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Reversed array:\n");
    for (int i = n - 1; i >= 0; i--) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd
 }
Enter the number of elements: 4
Enter the elements:
1
13
2
45
Reversed array:
45 2 13 1
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

8. Find the largest three distinct elements in an array: Input: arr[] = {10, 4, 3, 50, 23, 90} Output: 90, 50, 23



9. Move all zeroes to end of array



```c
#include <stdio.h>
void moveZeroesToEnd(int arr[], int n) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] != 0) {
            arr[count++] = arr[i];
        }
    }
    while (count < n) {
        arr[count++] = 0;
    }
}
int main() {
    int arr[] = {10, 0, 5, 0, 8, 0, 12};
    int n = sizeof(arr) / sizeof(arr[0]);

    moveZeroesToEnd(arr, n);
    printf("Array after moving zeroes to the end:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c
 }
Array after moving zeroes to the end:
10 5 8 12 0 0 0
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

10. Rearrange an array in maximum minimum form using Two Pointer Technique. Input: arr[] = {1, 2, 3, 4, 5, 6, 7} Output: arr[] = {7, 1, 6, 2, 5, 3, 4}.

10.c

```c
#include <stdio.h>
void rearrangeArray(int arr[], int n) {
    int result[n];
    int left = 0, right = n - 1, index = 0;

    while (left <= right) {
        if (index % 2 == 0) {
            result[index] = arr[right];
            right--;
        } else {
            result[index] = arr[left];
            left++;
        }
        index++;
    }
    for (int i = 0; i < n; i++) {
        arr[i] = result[i];
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    rearrangeArray(arr, n);
    printf("Rearranged array in maximum-minimum form:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

```
7 1 6 2 5 3 4
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

11. Print all Distinct (Unique) Elements in given Array:
    Input: arr[] = {12, 10, 9, 45, 2, 10, 10, 45} Output: 12, 10, 9, 2

11.c

```c
#include <stdio.h>

void printDistinctElements(int arr[], int n) {
    int count;
    for (int i = 0; i < n; i++) {
        count = 0;
        for (int j = 0; j < n; j++) {
            if (arr[i] == arr[j]) {
                count++;
            }
        }
        if (count == 1) {
            printf("%d ", arr[i]);
        }
    }
}

int main() {
    int arr[] = {12, 10, 9, 45, 2, 10, 10, 45};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Distinct elements in the array are:\n");
    printDistinctElements(arr, n);

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

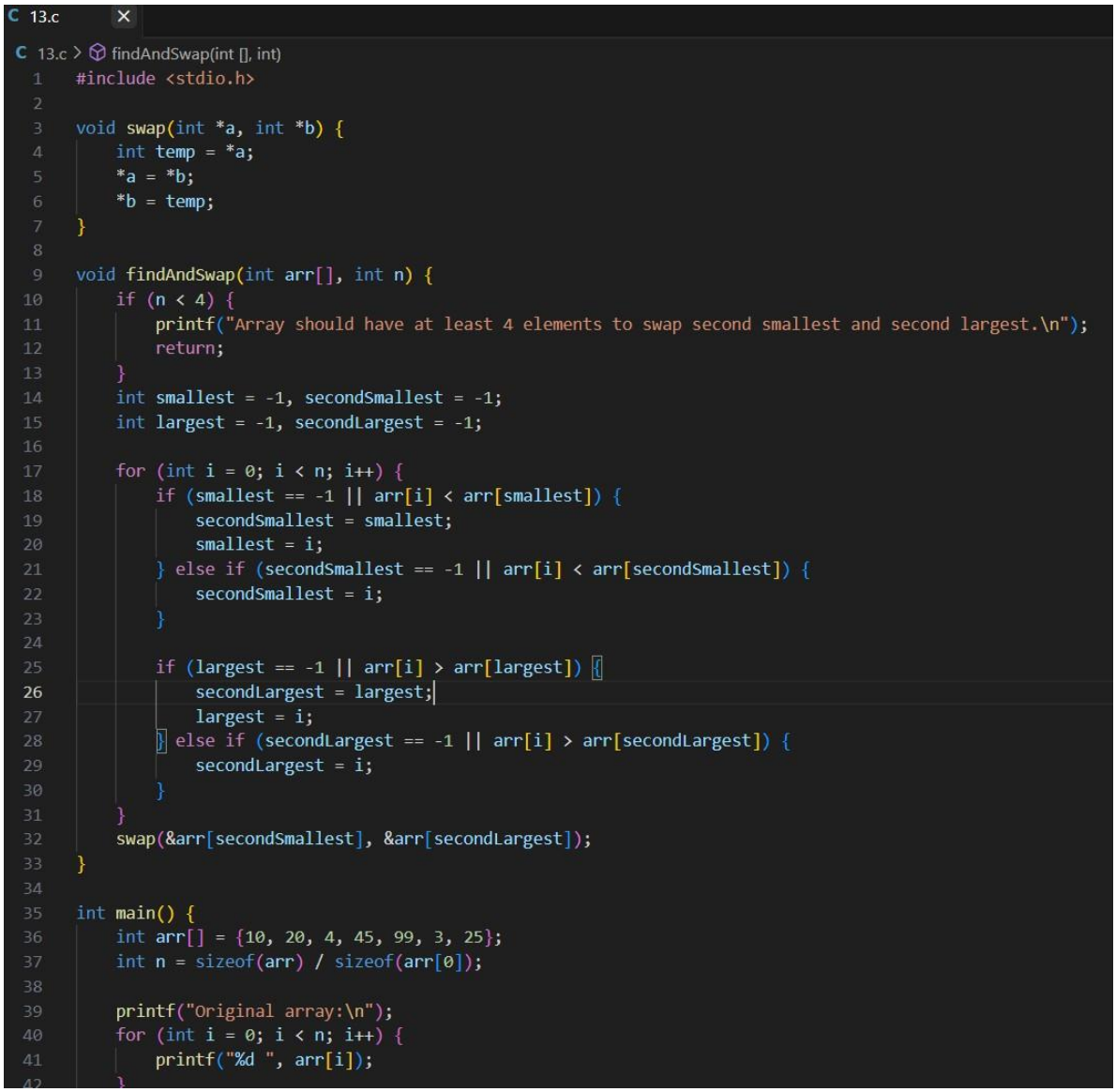
```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "
\11 }
Distinct elements in the array are:
12 9 2
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

12. Write a program to count the total number of nonzero elements in a two-dimensional array.

```c
#include <stdio.h>

int countNonZeroElements(int arr[][3], int rows, int cols) {
    int count = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (arr[i][j] != 0) {
                count++;
            }
        }
    }
    return count;
}
int main() {
    int arr[3][3] = {
        {1, 0, 3},
        {0, 2, 0},
        {4, 0, 5}
    };
    int rows = 3, cols = 3;

    int result = countNonZeroElements(arr, rows, cols);

    printf("Total number of non-zero elements: %d\n", result);

```

```
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems> cd "c:\Users\t
\12 }
Total number of non-zero elements: 5
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

13. Write a program using pointers to interchange the second biggest and the second smallest number in the array.



```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void findAndSwap(int arr[], int n) {
    if (n < 4) {
        printf("Array should have at least 4 elements to swap second smallest and second largest.\n");
        return;
    }
    int smallest = -1, secondSmallest = -1;
    int largest = -1, secondLargest = -1;

    for (int i = 0; i < n; i++) {
        if (smallest == -1 || arr[i] < arr[smallest]) {
            secondSmallest = smallest;
            smallest = i;
        } else if (secondSmallest == -1 || arr[i] < arr[secondSmallest]) {
            secondSmallest = i;
        }

        if (largest == -1 || arr[i] > arr[largest]) {
            secondLargest = largest;
            largest = i;
        } else if (secondLargest == -1 || arr[i] > arr[secondLargest]) {
            secondLargest = i;
        }
    }
    swap(&arr[secondSmallest], &arr[secondLargest]);
}

int main() {
    int arr[] = {10, 20, 4, 45, 99, 3, 25};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
```

```c
    printf("Original array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    findAndSwap(arr, n);
    printf("Array after swapping second smallest and second largest:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}

```

```
\13 }
Original array:
10 20 4 45 99 3 25
Array after swapping second smallest and second largest:
10 20 4 45 99 3 25
Array after swapping second smallest and second largest:
10 20 45 4 99 3 25
PS C:\Users\tilak\OneDrive\Desktop\Documents\C codes\Problems>
```

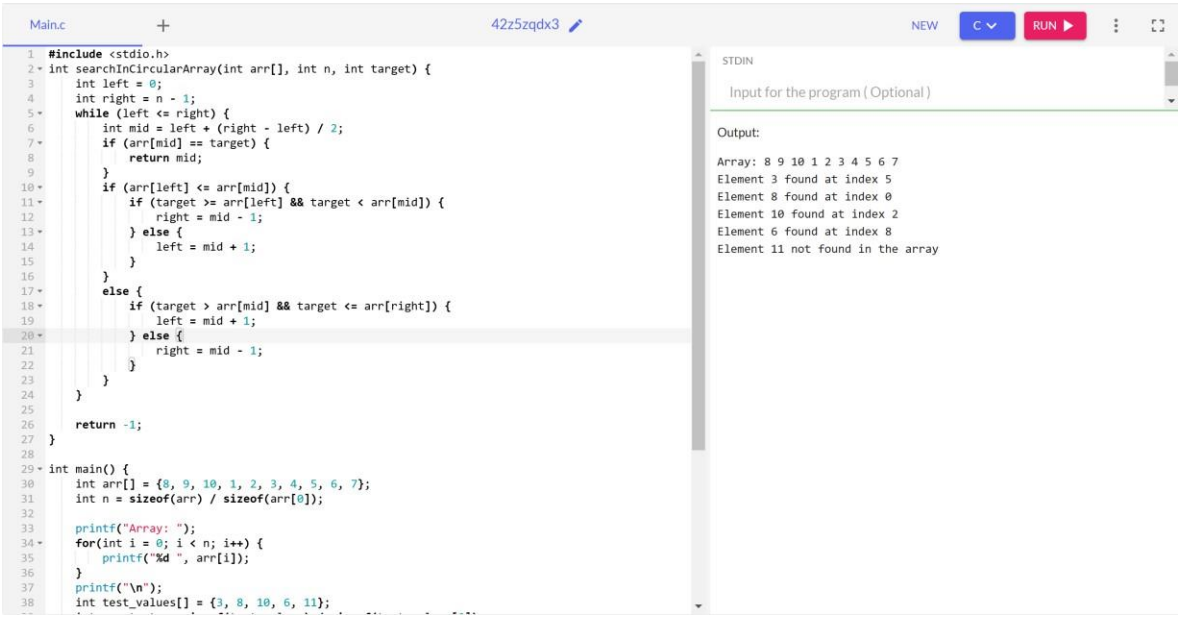
- LAB-2

1. FIND NUMBER OF ROTATIONS IN A CIRCULARLY SORTED ARRAY

```c
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void findSecondLargestAndSecondSmallest(int arr[], int n) {
    if (n < 4) {
        printf("Array must have at least four elements.\n");
        return;
    }
    int *secondLargest = NULL, *secondSmallest = NULL;
    int largest = arr[0], smallest = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > largest) {
            secondLargest = &largest;
            largest = arr[i];
        } else if (arr[i] < smallest) {
            secondSmallest = &smallest;
            smallest = arr[i];
        }
    }
    for (int i = 0; i < n; i++) {
        if (arr[i] < largest && (secondLargest == NULL || arr[i] > *secondLargest)) {
            secondLargest = &arr[i];
        }
        if (arr[i] > smallest && (secondSmallest == NULL || arr[i] < *secondSmallest)) {
            secondSmallest = &arr[i];
        }
    }
    swap(secondLargest, secondSmallest);
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

```
Output:

Original array:
12 35 1 10 34 1 3
Array after interchanging second largest and second smallest:
12 35 1 10 34 1 3
```

2. SEARCH AN ELEMENT IN A CRCULARLY SORTED ARRAY



```c
#include <stdio.h>
int searchInCircularArray(int arr[], int n, int target) {
    int left = 0;
    int right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid;
        }
        if (arr[left] <= arr[mid]) {
            if (target >= arr[left] && target < arr[mid]) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }
        else {
            if (target > arr[mid] && target <= arr[right]) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
    }

    return -1;
}

int main() {
    int arr[] = {8, 9, 10, 1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Array: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    int test_values[] = {3, 8, 10, 6, 11};
```

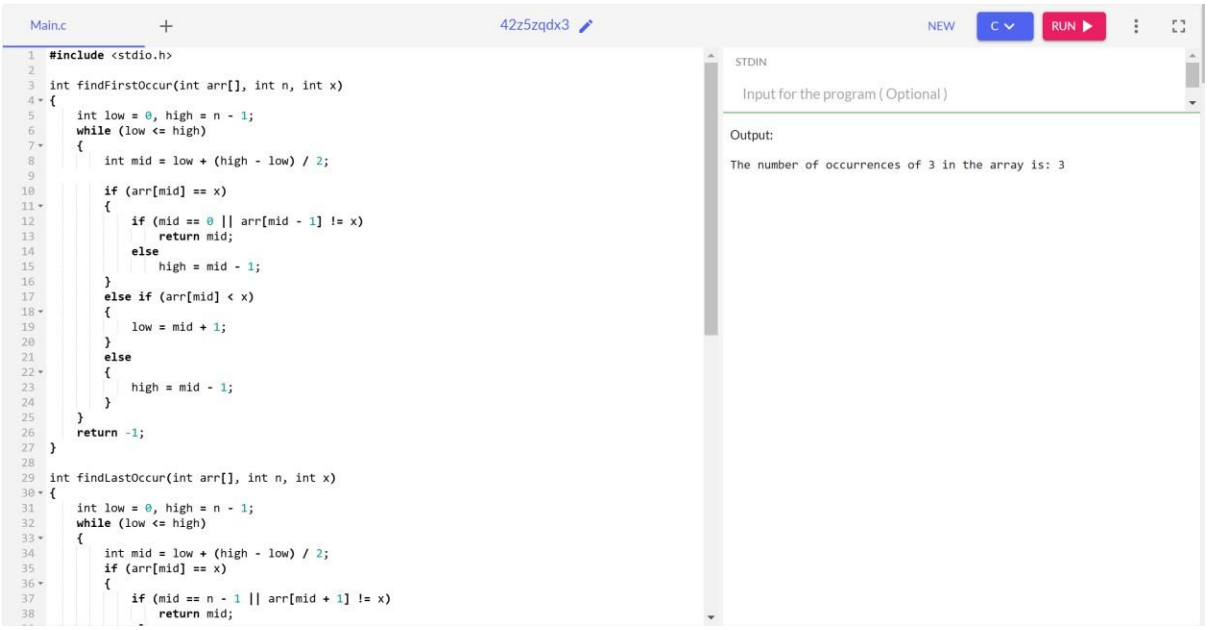
```
Output:

Array: 8 9 10 1 2 3 4 5 6 7
Element 3 found at index 5
Element 8 found at index 0
Element 10 found at index 2
Element 6 found at index 8
Element 11 not found in the array
```

3. Find the first or last occurrence of a given number in a sorted array

```c
#include <stdio.h>

int findFirstOccur(int arr[], int n, int x)
{
    int low = 0, high = n - 1;

    while (low <= high)
    {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x)
        {
            if (mid == 0 || arr[mid - 1] != x)
                return mid;
            else
                high = mid - 1;
        }
        else if (arr[mid] < x)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    return -1;
}

int findLastOccur(int arr[], int n, int x)
{
    int low = 0, high = n - 1;

    while (low <= high)
    {
        int mid = low + (high - low) / 2;

        if (arr[mid] == x)
        {
```

```
STDIN
Input for the program (Optional)

Output:

First occurrence of 3 is at index 3
Last occurrence of 3 is at index 5
```

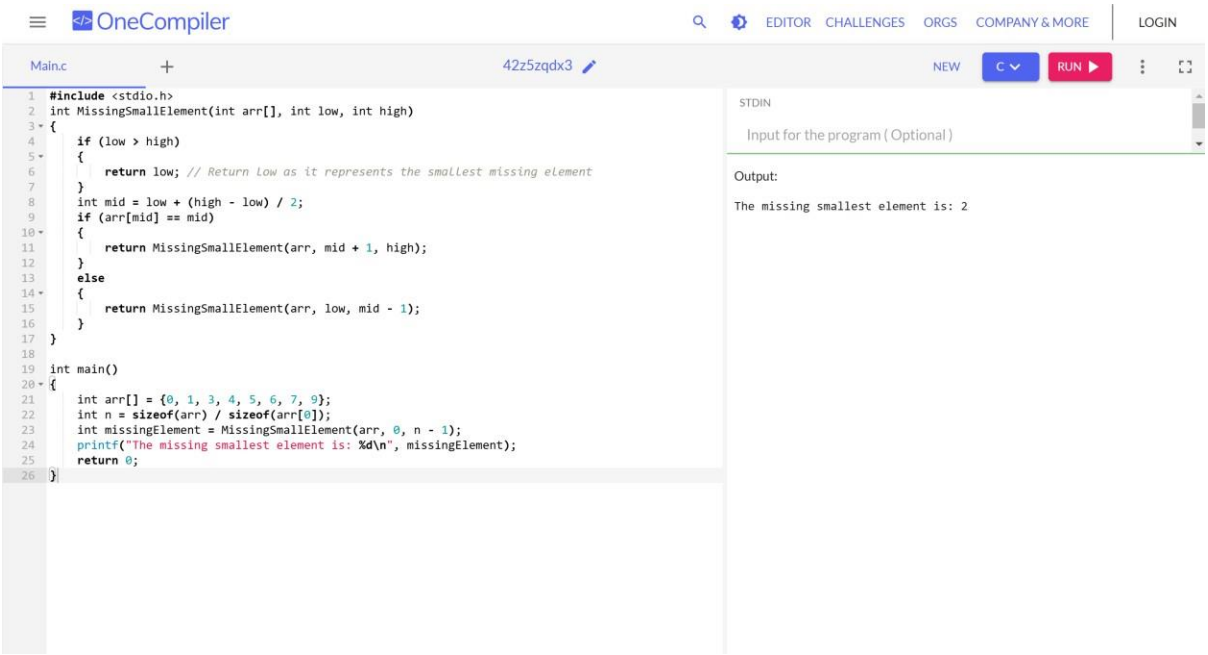
4. Count occurences of a given number in a sorted array



```c
#include <stdio.h>

int findFirstOccur(int arr[], int n, int x)
{
    int low = 0, high = n - 1;
    while (low <= high)
    {
        int mid = low + (high - low) / 2;

        if (arr[mid] == x)
        {
            if (mid == 0 || arr[mid - 1] != x)
                return mid;
            else
                high = mid - 1;
        }
        else if (arr[mid] < x)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    return -1;
}

int findLastOccur(int arr[], int n, int x)
{
    int low = 0, high = n - 1;
    while (low <= high)
    {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x)
        {
            if (mid == n - 1 || arr[mid + 1] != x)
                return mid;
```

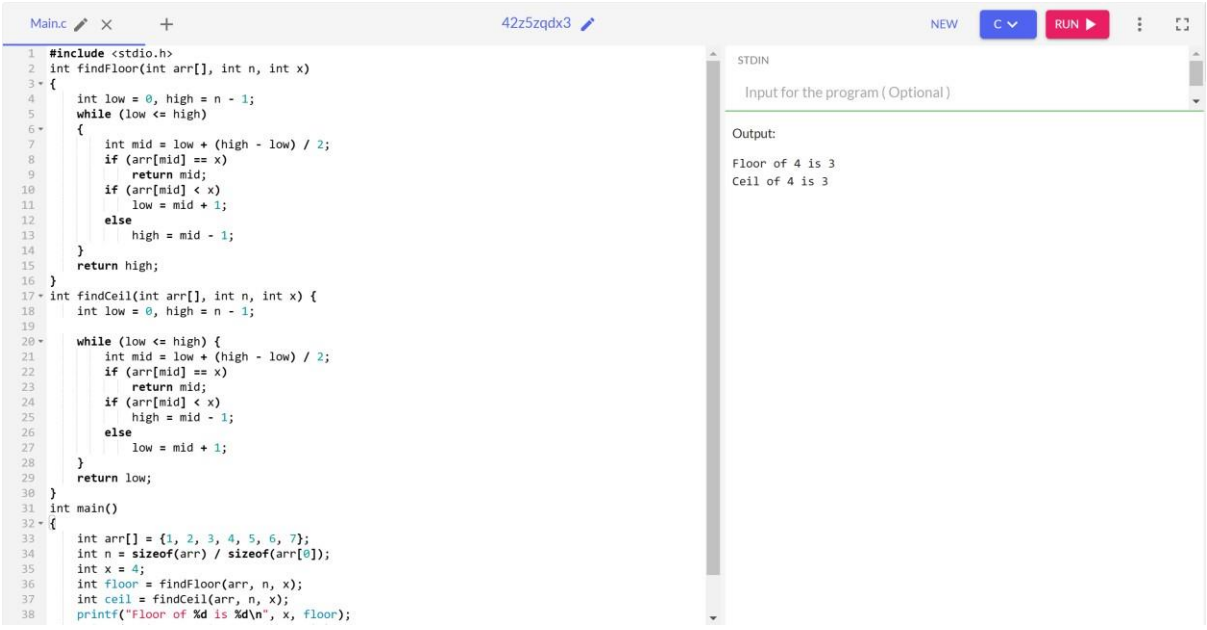
```
STDIN
Input for the program (Optional)

Output:

The number of occurrences of 3 in the array is: 3
```

5. Find the smallest missing element from a sorted array



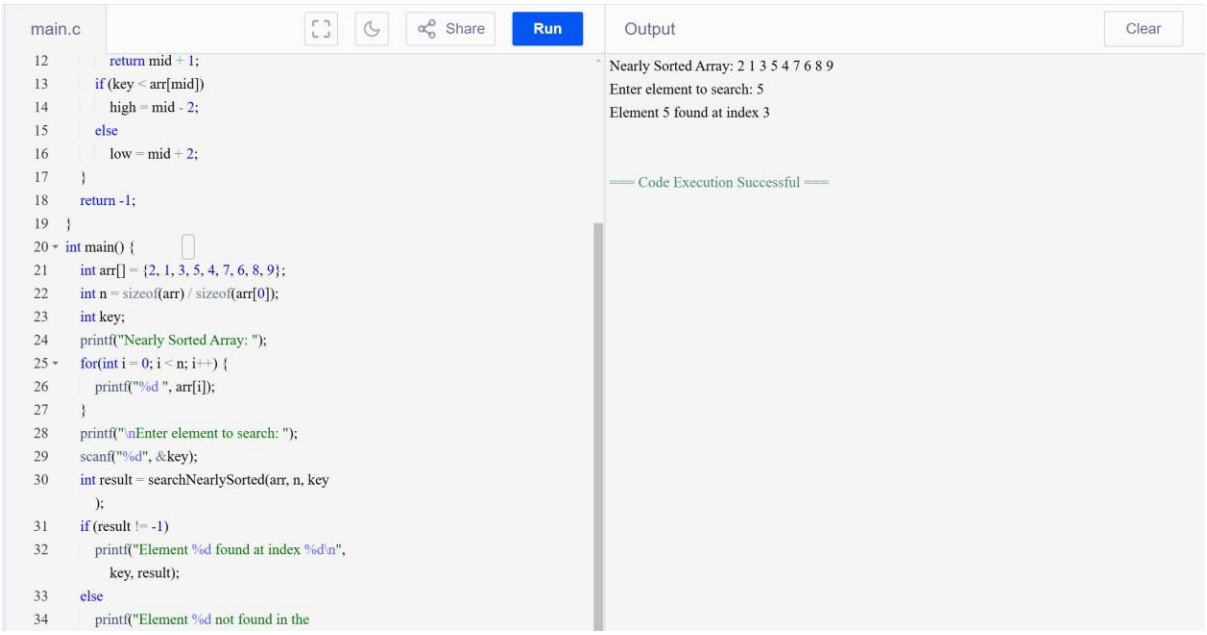

```c
#include <stdio.h>
int MissingSmallElement(int arr[], int low, int high)
{
    if (low > high)
    {
        return low; // Return low as it represents the smallest missing element
    }
    int mid = low + (high - low) / 2;
    if (arr[mid] == mid)
    {
        return MissingSmallElement(arr, mid + 1, high);
    }
    else
    {
        return MissingSmallElement(arr, low, mid - 1);
    }
}

int main()
{
    int arr[] = {0, 1, 3, 4, 5, 6, 7, 9};
    int n = sizeof(arr) / sizeof(arr[0]);
    int missingElement = MissingSmallElement(arr, 0, n - 1);
    printf("The missing smallest element is: %d\n", missingElement);
    return 0;
}
```

6. Find floor and ceil of a number in a sorted integer array

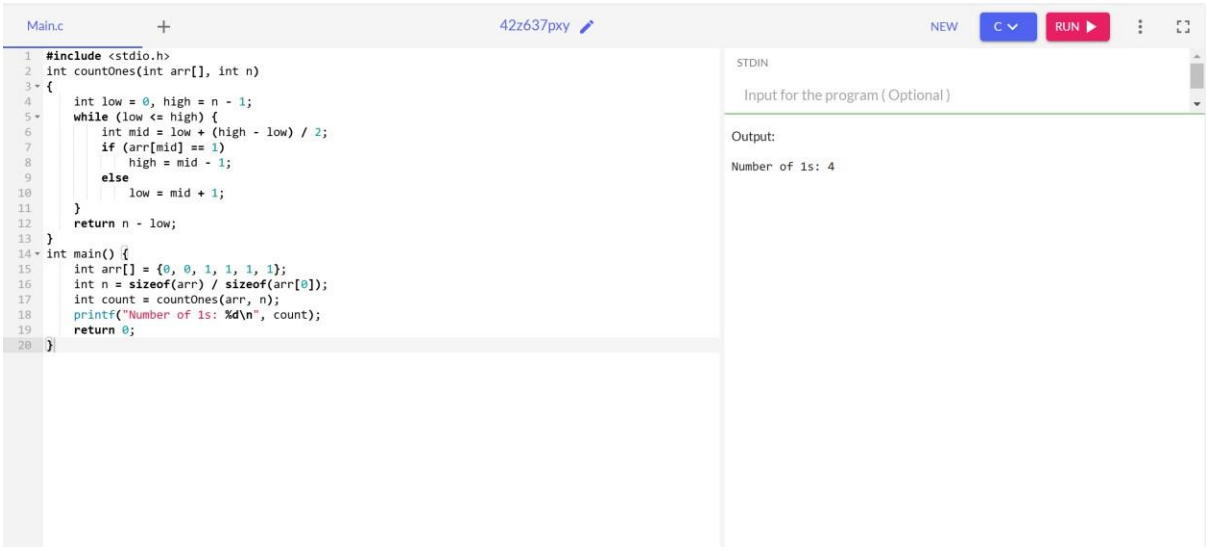



```c
#include <stdio.h>
int findFloor(int arr[], int n, int x)
{
    int low = 0, high = n - 1;
    while (low <= high)
    {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] < x)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return high;
}
int findCeil(int arr[], int n, int x) {
    int low = 0, high = n - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] < x)
            high = mid - 1;
        else
            low = mid + 1;
    }
    return low;
}
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 4;
    int floor = findFloor(arr, n, x);
    int ceil = findCeil(arr, n, x);
    printf("Floor of %d is %d\n", x, floor);
```

7. Search in a nearly sorted array in logarithmic time



8. Find the number of 1's in a sorted binary array

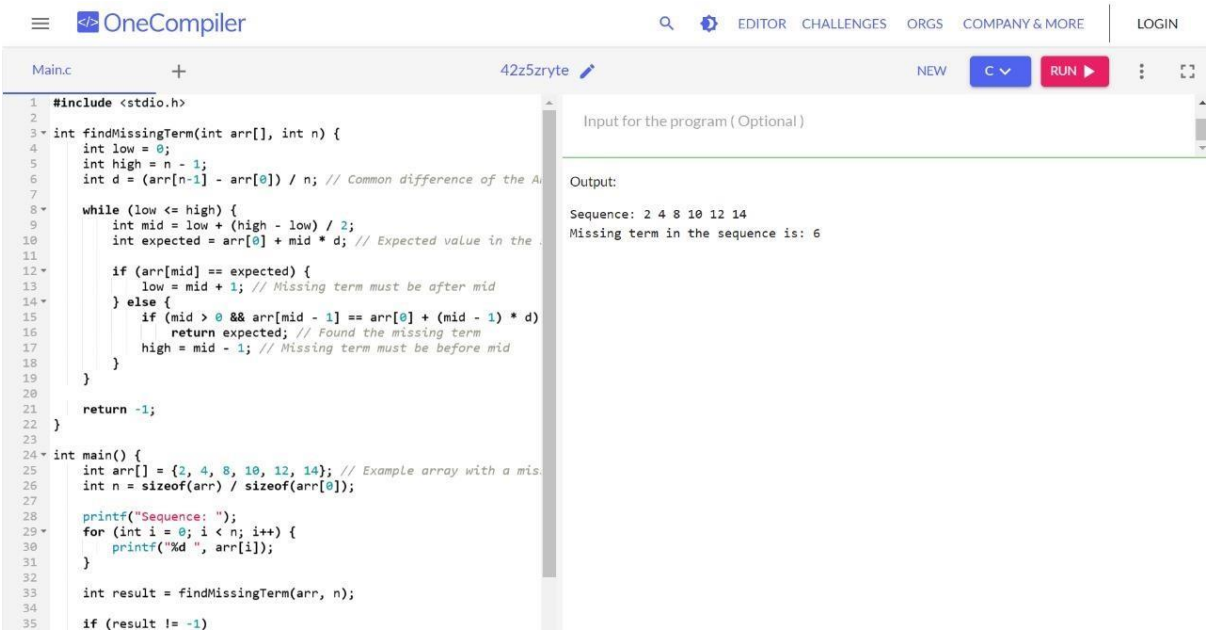


9. Find the peak element in an array

```c
#include <stdio.h>
int findPeakElement(int arr[], int n) {
    // If array has only one element, it's the peak
    if (n == 1)
        return 0;
    if (arr[0] >= arr[1])
        return 0;
    if (arr[n-1] >= arr[n-2])
        return n-1;
    int low = 1;
    int high = n - 2;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] >= arr[mid-1] && arr[mid] >= arr[mid+1])
            return mid;
        if (arr[mid-1] > arr[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1;
}

int main() {
    int arr[] = {1, 3, 20, 4, 1, 0};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Array: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    int result = findPeakElement(arr, n);
    if (result != -1)
        printf("\nPeak element is %d at index %d\n", arr[result], result);
    else
        printf("\nNo peak element found\n");
    return 0;
}
```

```
Output:

Array: 1 3 20 4 1 0
Peak element is 20 at index 2
```

10. Find the missing term in a sequence in logarithmic time



```c
#include <stdio.h>

int findMissingTerm(int arr[], int n) {
    int low = 0;
    int high = n - 1;
    int d = (arr[n-1] - arr[0]) / n; // Common difference of the A

    while (low <= high) {
        int mid = low + (high - low) / 2;
        int expected = arr[0] + mid * d; // Expected value in the

        if (arr[mid] == expected) {
            low = mid + 1; // Missing term must be after mid
        } else {
            if (mid > 0 && arr[mid - 1] == arr[0] + (mid - 1) * d)
                return expected; // Found the missing term
            high = mid - 1; // Missing term must be before mid
        }
    }

    return -1;
}

int main() {
    int arr[] = {2, 4, 8, 10, 12, 14}; // Example array with a mis
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Sequence: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    int result = findMissingTerm(arr, n);

    if (result != -1)
```

```
Output:

Sequence: 2 4 8 10 12 14
Missing term in the sequence is: 6
```

11. Find floor and ceil of a number in a sorted array(recursive solution)

main.c Share Run Output Clear

```c
#include <stdio.h>
int findFloor(int arr[], int low, int high, int x) {
    if (low > high)
        return -1;
    if (x >= arr[high])
        return arr[high];
    if (x < arr[low])
        return -1;
    int mid = low + (high - low) / 2;
    if (arr[mid] == x)
        return arr[mid];
    if (mid > 0 && arr[mid-1] <= x && x < arr[mid])
        return arr[mid-1];
    if (x < arr[mid])
        return findFloor(arr, low, mid-1, x);
    return findFloor(arr, mid+1, high, x);
}
int findCeil(int arr[], int low, int high, int x) {
    if (low > high)
        return -1;
    if (x <= arr[low])
        return arr[low];
```

```
Sorted Array: 1 2 4 6 8 10 12 14
Enter number to find floor and ceil: 5.5
Floor of 5 is 4
Ceil doesn't exist

=== Code Execution Successful ===
```

12. Find frequency of each element in a sorted array containing duplicates

Main.c 42z637pxy NEW C RUN

```c
#include <stdio.h>
int findFirst(int arr[], int low, int high, int x) {
    if (low > high)
        return -1;

    int mid = low + (high - low) / 2;

    if ((mid == 0 || x > arr[mid-1]) && arr[mid] == x)
        return mid;
    else if (x > arr[mid])
        return findFirst(arr, mid+1, high, x);
    else
        return findFirst(arr, low, mid-1, x);
}
int findLast(int arr[], int low, int high, int x) {
    if (low > high)
        return -1;
    int mid = low + (high - low) / 2;
    if ((mid == high || x < arr[mid+1]) && arr[mid] == x)
        return mid;
    else if (x < arr[mid])
        return findLast(arr, low, mid-1, x);
    else
        return findLast(arr, mid+1, high, x);
}
void findFrequency(int arr[], int n) {
    int i = 0;
    while (i < n) {
        int first = findFirst(arr, 0, n-1, arr[i]);
        int last = findLast(arr, 0, n-1, arr[i]);

        if (first != -1) {
            printf("Element %d occurs %d times\n",
                    arr[i], last - first + 1);
            i = last + 1;
        } else {
            i++;
        }
```

STDIN

Input for the program (Optional)

Output:

```
Sorted Array: 1 1 2 2 2 2 3 4 4 4 5

Frequency Count:
Element 1 occurs 2 times
Element 2 occurs 4 times
Element 3 occurs 1 times
Element 4 occurs 3 times
Element 5 occurs 1 times
```

13. Find square root of a number using binary search

main.c Share Run Output Clear

```c
#include <stdio.h>
double findSquareRoot(int x, int precision) {
    if (x == 0 || x == 1)
        return x;
    double low = 1;
    double high = x;
    double epsilon = 1.0;
    for(int i = 0; i < precision; i++) {
        epsilon /= 10;
    }
    double result = 0;
    while (high - low > epsilon) {
        double mid = (low + high) / 2;
        double sqr = mid * mid;
        if (sqr == x) {
            return mid;
        }
        else if (sqr < x) {
            low = mid;
            result = mid;
        }
        else {
            high = mid;
        }
    }

    return result;
```

```
Enter number to find square root: 4
Enter precision (number of decimal places): 2
Square root of 4 is approximately 1.9960937500


=== Code Execution Successful ===
```

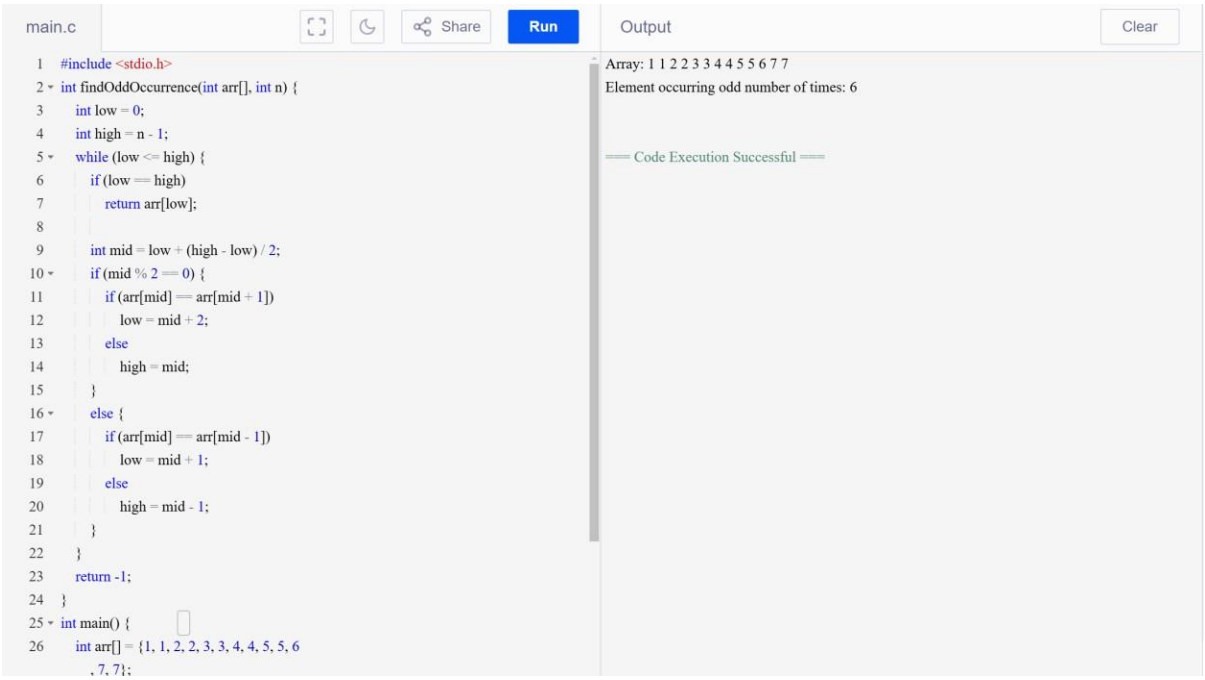
14. Divison of two numbers using binary search algorithm

main.c Share Run Output Clear

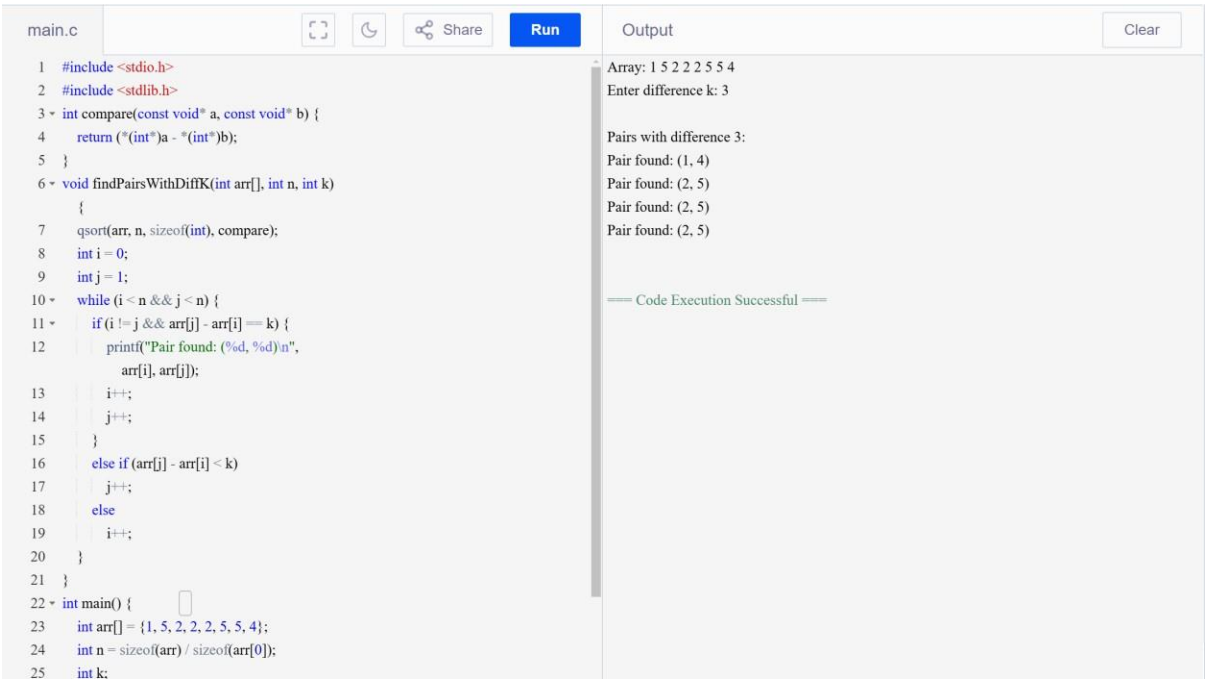
```c
#include <stdio.h>
#include <stdlib.h>
double divide(int dividend, int divisor, int
    precision) {
    if (divisor == 0) {
        printf("Error: Division by zero\n");
        exit(1);
    }
    int sign = ((dividend < 0) ^ (divisor < 0))
        ? -1 : 1;
    dividend = abs(dividend);
    divisor = abs(divisor);
    int quotient = 0;
    int low = 0;
    int high = dividend;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (mid * divisor == dividend) {
            quotient = mid;
            break;
        }
        else if (mid * divisor < dividend) {
            quotient = mid;
            low = mid + 1;
        }
        else {
            high = mid - 1;
```

```
Enter dividend: 128
Enter divisor: 2
Enter precision (decimal places): 2
128 / 2 = 64.0000000000


=== Code Execution Successful ===
```

15. Find the odd occuring element in an array in logarithmic time



16. Find pairs with differene ‘k’ in an array |constant space solution



17. Find ‘k’ closest elements to a given value in an array

main.c

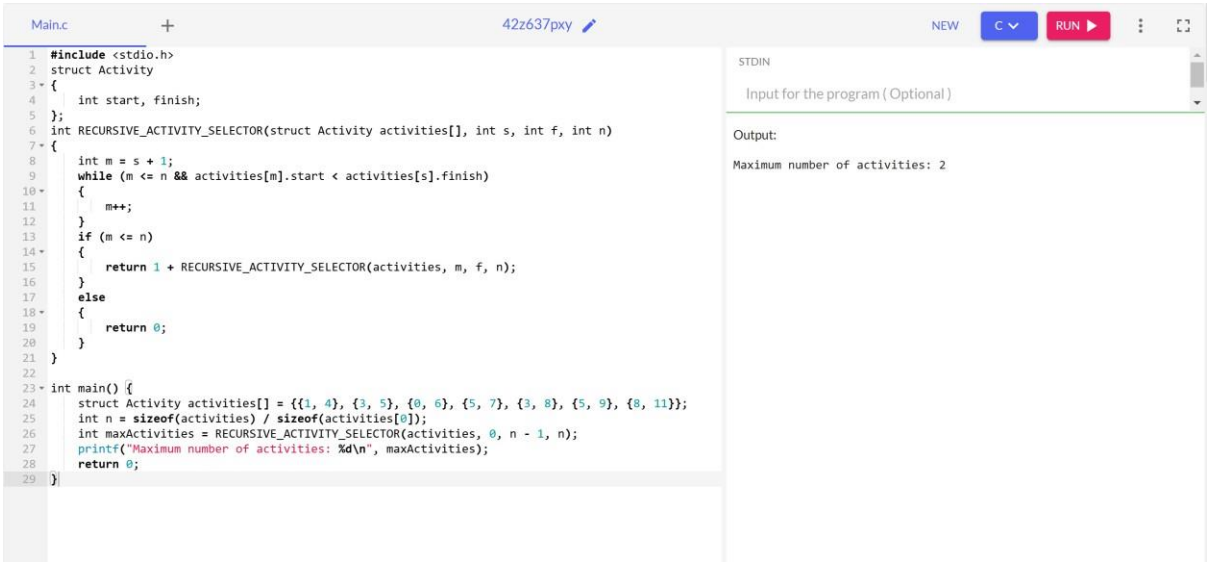
Share Run Output Clear

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int findCrossOver(int arr[], int n, int x) {
    int low = 0;
    int high = n - 1;
    if (x >= arr[high])
        return high;
    if (x < arr[low])
        return low;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] <= x && arr[mid + 1] > x)
            return mid;
        else if (arr[mid] < x)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
void printKClosest(int arr[], int n, int k, int
    x) {
    int left = findCrossOver(arr, n, x);
    int right = left + 1;
    if (arr[left] == x)
        left--;
```

```
Array: 12 16 22 30 35 39 42 45 48 50 53 55 56
Enter value x: 40
Enter number of closest elements (k): 5
5 closest elements to 40 are: 39 42 35 45 48


=== Code Execution Successful ===
```

- LAB-3

1. Implement the activity selection problem to get a clear understanding of greedy approach.



2. Get a detailed insight of dynamic programming approach by the implementation of Matrix Chain Multiplication problem and see the impact of parenthesis positioning on time requirements for matrix multiplication.

main.c

```c
#include <stdio.h>
#include <limits.h>
int matrixChainOrder(int p[], int n) {
    int m[n][n];
    int s[n][n];
    for (int i = 1; i < n; i++)
        m[i][i] = 0;
    for (int L = 2; L < n; L++) {
        for (int i = 1; i < n - L + 1; i++) {
            int j = i + L - 1;
            m[i][j] = INT_MAX;
            for (int k = i; k <= j - 1; k++) {
                int q = m[i][k] + m[k + 1][j] +
                    p[i - 1] * p[k] * p[j];
                if (q < m[i][j]) {
                    m[i][j] = q;
                    s[i][j] = k;
                }
            }
        }
    }
    return m[1][n - 1];
}
void printOptimalParenthesis(int s[][10], int i,
    int j) {
    if (i == j)
        printf("A%d", i);
```

Output

```
Minimum number of multiplications is 18
Optimal Parenthesization is: ((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((

=== Code Exited With Errors ===
```

3. Compare the performance of Dijkstra and Bellman ford algorithm for the single source shortest path problem.

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <time.h>

#define V 6  // Number of vertices
#define INF INT_MAX

// Structure to represent an edge
struct Edge {
    int src, dest, weight;
};

// Function to initialize the distance array
void initializeDistances(int dist[], int src) {
    for (int i = 0; i < V; i++)
        dist[i] = INF;
    dist[src] = 0;
}

// Function to find the vertex with minimum distance
int minDistance(int dist[], int visited[]) {
    int min = INF, min_index;
    for (int v = 0; v < V; v++) {
        if (!visited[v] && dist[v] <= min) {
            min = dist[v];
            min_index = v;
        }
    }
    return min_index;
}

// Dijkstra's Algorithm
void dijkstra(int graph[V][V], int src, int dist[]) {
    int visited[V];
```

STDIN

Input for the program (Optional)

Output:

```
Dijkstra's Algorithm Results:
Vertex   Distance from Source
0                0
1                4
2                12
3                19
4                26
5                16
Time taken: 0.000002 seconds

Bellman-Ford Algorithm Results:
Vertex   Distance from Source
0                0
1                4
2                12
3                19
4                26
5                16
Time taken: 0.000001 seconds
```

4. Through 0/1 Knapsack problem, analyze the greedy and dynamic programming approach for the same dataset.

```c
#include <stdio.h>
typedef struct {
    int weight;
    int value;
    float ratio;
} Item;
int compare(const void *a, const void *b) {
    Item *item1 = (Item *)a;
    Item *item2 = (Item *)b;
    return (item2->ratio > item1->ratio) - (item2->ratio < item1->rat
}
float greedyKnapsack(Item items[], int n, int capacity) {
    qsort(items, n, sizeof(Item), compare);
    float totalValue = 0.0;
    for (int i = 0; i < n; i++) {
        if (items[i].weight <= capacity) {
            capacity -= items[i].weight;
            totalValue += items[i].value;
        } else {
            totalValue += items[i].value * ((float)capacity / items[
            break;
        }
    }
    return totalValue;
}
int main() {
    Item items[] = {{10, 60}, {20, 100}, {30, 120}};
    int n = sizeof(items) / sizeof(items[0]);
    int capacity = 50;

    float maxValue = greedyKnapsack(items, n, capacity);
    printf("Maximum value in Greedy Knapsack: %.2f\n", maxValue);
    return 0;
}
```

STDIN

Input for the program (Optional)

Output:

```
Maximum value in Greedy Knapsack: 240.00
```

```c
#include <stdio.h>
int knapsackDP(int capacity, int weights[], int values[], int n) {
    int dp[n + 1][capacity + 1];
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (weights[i - 1] <= w)
                dp[i][w] = (values[i - 1] + dp[i - 1][w - weights[i -
                            values[i - 1] + dp[i - 1][w - weights[i -
            else
                dp[i][w] = dp[i - 1][w];
        }
    }
    return dp[n][capacity];
}
int main() {
    int values[] = {60, 100, 120};
    int weights[] = {10, 20, 30};
    int capacity = 50;
    int n = sizeof(values) / sizeof(values[0]);
    int maxValue = knapsackDP(capacity, weights, values, n);
    printf("Maximum value in Dynamic Programming Knapsack: %d\n", max
    return 0;
}
```

STDIN

Input for the program (Optional)

Output:

```
Maximum value in Dynamic Programming Knapsack: 220
```

5. Implement the sum of subset and N Queen problem.

```c
#include <stdio.h>
int isSubsetSum(int set[], int n, int sum) {
    if (sum == 0)
        return 1;
    if (n == 0)
        return 0;

    if (set[n - 1] > sum)
        return isSubsetSum(set, n - 1, sum);

    return isSubsetSum(set, n - 1, sum) || isSubsetSum(set, n - 1, su
}
int main() {
    int set[] = {3, 34, 4, 12, 5, 2};
    int sum = 9;
    int n = sizeof(set) / sizeof(set[0]);

    if (isSubsetSum(set, n, sum))
        printf("Found a subset with given sum\n");
    else
        printf("No subset with given sum\n");

    return 0;
}
```

STDIN

Input for the program (Optional)

Output:

```
Found a subset with given sum
```

main.c

Share | Run | Output | Clear

```
}
int main() {
    int N;
    printf("Enter the size of the chessboard (N
        ): ");
    scanf("%d", &N);
    if (N <= 0) {
        printf("Invalid board size!\n");
        return 1;
    }
    if (N == 2 || N == 3) {
        printf("Solution does not exist for N =
            %d\n", N);
        return 1;
    }
    int **board = initializeBoard(N);
    if (solveNQueens(board, 0, N)) {
        printf("\nSolution exists for N = %d", N
            );
        printBoard(board, N);
    } else {
        printf("\nSolution does not exist for N
            = %d\n", N);
    }
    freeBoard(board, N);
    return 0;
}
```

```
Enter the size of the chessboard (N): 5

Solution exists for N = 5
Solution Board:
Q . . . .
. . . Q .
. Q . . .
. . . . Q
. . Q . .


=== Code Execution Successful ===
```

6. Compare the Backtracking and Branch & Bound Approach by the implementation of 0/1 Knapsack problem. Also compare the performance with dynamic programming approach.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>

// Structure for Branch and Bound
typedef struct {
    float bound;
    float profit;
    float weight;
    int level;
} Node;

// Global variables to store best solution for
    backtracking
int* bestSolution;
int maxProfit = 0;

// Function to get maximum of two integers
int max(int a, int b) {
    return (a > b) ? a : b;
```

```
Enter number of items: 34
Enter knapsack capacity: 25

Generated Test Data:
Weights: 34 37 38 18 5 13 38 4 35 1 24 11 35 18 22 10 49 12
  32 29 30 45 38 30 1 8 32 16 20 40 45 45 47 30
Values: 35 6 34 92 100 5 97 60 4 83 1 90 48 72 12 55 97 86
  29 89 91 69 48 24 30 27 20 67 83 91 49 93 8 31

Comparing different approaches:

Time taken by Dynamic Programming: 0.000016 seconds
Time taken by Backtracking: 0.000039 seconds
Time taken by Branch and Bound: 0.000008 seconds

Results:
Dynamic Programming: 363
Backtracking: 363
Branch and Bound: 305
```