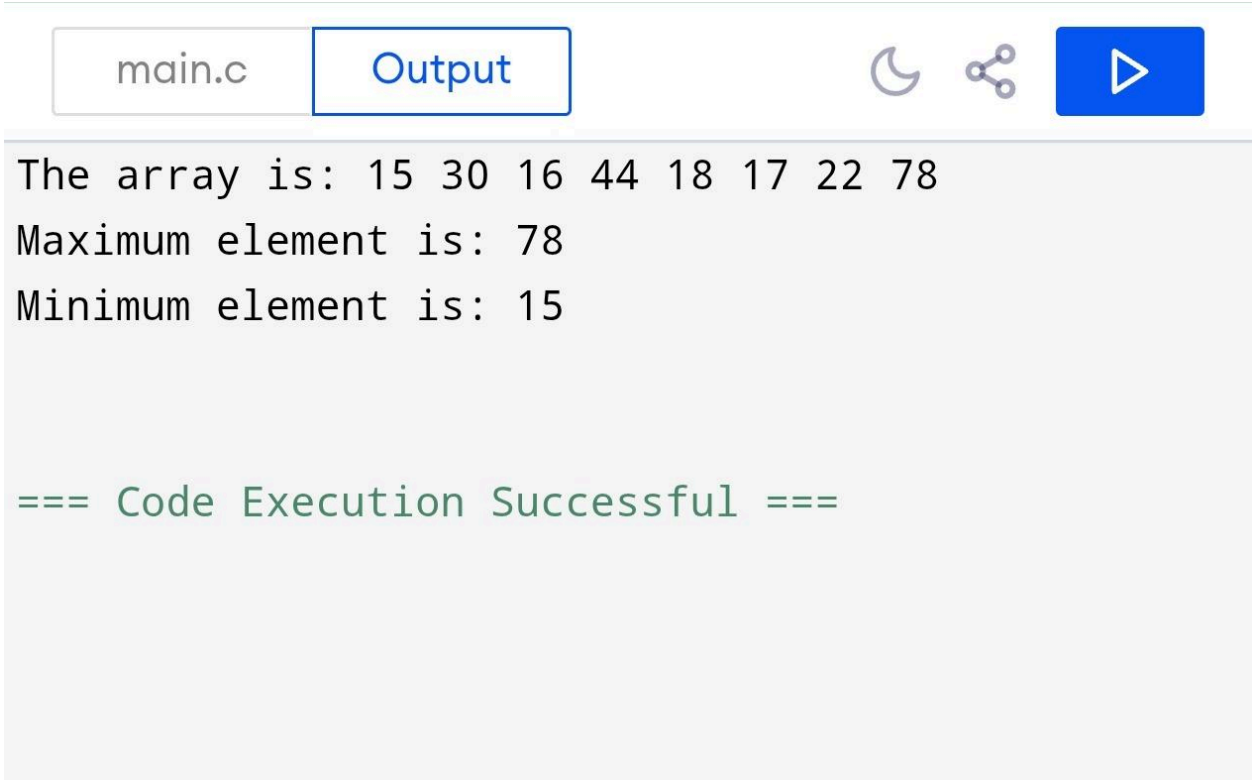
1. **Finding max and min element in array using divide and conquer method**

```
#include <stdio.h>

typedef struct {
    int min;
    int max;
} MinMaxResult;

MinMaxResult findMaxMin(int arr[], int low, int high) {
    MinMaxResult result, leftResult, rightResult;
    int mid;

    if (low == high) {
        result.max = arr[low];
        result.min = arr[low];
        return result;
    }

    if (high - low == 1) {
        if (arr[low] > arr[high]) {
            result.max = arr[low];
            result.min = arr[high];
        } else {
            result.max = arr[high];
            result.min = arr[low];
        }
        return result;
    }

    mid = (low + high) / 2;

    leftResult = findMaxMin(arr, low, mid);

    rightResult = findMaxMin(arr, mid + 1, high);

    if (leftResult.max > rightResult.max) {
        result.max = leftResult.max;
    } else {
        result.max = rightResult.max;
    }

    if (leftResult.min < rightResult.min) {
```

```
        result.min = leftResult.min;
    } else {
        result.min = rightResult.min;
    }

    return result;
}

int main() {
    int arr[] = {15,30,16,44,18,17,22,78};
    int n = sizeof(arr) / sizeof(arr[0]);
    MinMaxResult result = findMaxMin(arr, 0, n - 1);

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

    printf("Maximum element is: %d\n", result.max);
    printf("Minimum element is: %d\n", result.min);

    return 0;
}
```



2. **Hash table**

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

typedef struct Node {
    int key;
    int value;
    struct Node* next;
} Node;

#define TABLE_SIZE 10
typedef struct {
    Node* buckets[TABLE_SIZE];
} HashTable;

int hash(int key) {
    return key % TABLE_SIZE;
}

Node* createNode(int key, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    if (newNode == NULL) {
        printf("Error: Memory allocation failed!\n");
        exit(1);
    }
    newNode->key = key;
    newNode->value = value;
    newNode->next = NULL;
    return newNode;
}

void initializeTable(HashTable* ht) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        ht->buckets[i] = NULL;
    }
}

void insert(HashTable* ht, int key, int value) {
    int index = hash(key);
    Node* newNode = createNode(key, value);

    if (ht->buckets[index] == NULL) {
        ht->buckets[index] = newNode;
    } else {
        newNode->next = ht->buckets[index];
        ht->buckets[index] = newNode;
    }
    printf("Successfully inserted key %d with value %d.\n", key, value);
}

void search(HashTable* ht, int key) {
    int index = hash(key);
    Node* current = ht->buckets[index];

    while (current != NULL) {
        if (current->key == key) {
            printf("Key %d found with value %d.\n", key, current->value);
            return;
        }
        current = current->next;
    }
    printf("Key %d not found in the hash table.\n", key);
}

void delete(HashTable* ht, int key) {
```

```
    int index = hash(key);
    Node* current = ht->buckets[index];
    Node* prev = NULL;

    while (current != NULL && current->key != key) {
        prev = current;
        current = current->next;
    }

    if (current == NULL) {
        printf("Key %d not found for deletion.\n", key);
        return;
    }

    if (prev == NULL) {
        ht->buckets[index] = current->next;
    } else {
        prev->next = current->next;
    }

    free(current);
    printf("Successfully deleted key %d.\n", key);
}

void display(HashTable* ht) {
    printf("\nHash Table Contents:\n");
    for (int i = 0; i < TABLE_SIZE; i++) {
        printf("Bucket %d:", i);
        Node* current = ht->buckets[i];
        while (current != NULL) {
            printf(" -> (%d, %d)", current->key, current->value);
            current = current->next;
        }
        printf(" -> NULL\n");
    }
    printf("\n");
}

int main() {
    HashTable ht;
    initializeTable(&ht);

    int choice, key, value;
```

```
    do {
        printf("\nHash Table Operations Menu:\n");
        printf("1. Insert\n");
        printf("2. Search\n");
        printf("3. Delete\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter key to insert: ");
                scanf("%d", &key);
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(&ht, key, value);
                break;
            case 2:
                printf("Enter key to search: ");
                scanf("%d", &key);
                search(&ht, key);
                break;
            case 3:
                printf("Enter key to delete: ");
                scanf("%d", &key);
                delete(&ht, key);
                break;
            case 4:
                display(&ht);
                break;
            case 5:
                printf("Exiting program. Goodbye!\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

```
Enter key to insert: 1
Enter value to insert: 12
Successfully inserted key 1 with value 12.

Hash Table Operations Menu:
1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 4

Hash Table Contents:
Bucket 0: -> NULL
Bucket 1: -> (1, 12) -> NULL
Bucket 2: -> NULL
Bucket 3: -> NULL
Bucket 4: -> NULL
Bucket 5: -> NULL
Bucket 6: -> NULL
Bucket 7: -> NULL
Bucket 8: -> NULL
Bucket 9: -> NULL
```