1 Write a c program for priority queue.

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    int priority;
    struct Node* next;
};
struct Node* newNode(int data, int priority) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = data;
    temp->priority = priority;
    temp->next = NULL;
    return temp;
}
int isEmpty(struct Node** head) {
    return (*head) == NULL;
}
void push(struct Node** head, int data, int priority) {
    struct Node* start = (*head);
    struct Node* temp = newNode(data, priority);
    if (isEmpty(head) || (*head)->priority > priority) {
        temp->next = *head;
        *head = temp;
    } else {
        while (start->next != NULL && start->next->priority <= priority) {
            start = start->next;
        }
        temp->next = start->next;
        start->next = temp;
    }
}
void pop(struct Node** head) {
    if (isEmpty(head)) {
        printf("Priority Queue is empty\n");
        return;
    }
    struct Node* temp = *head;
    (*head) = (*head)->next;
    free(temp);
}
int peek(struct Node** head) {
    if (isEmpty(head)) {
        printf("Priority Queue is empty\n");
        return -1;
    }
    return (*head)->data;
}
void display(struct Node* head) {
    if (isEmpty(&head)) {
        printf("Priority Queue is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("Data: %d Priority: %d\n", temp->data, temp->priority);
        temp = temp->next;
    }
}
int main() {
    struct Node* pq = NULL;
    push(&pq, 4, 1);
    push(&pq, 5, 2);
    push(&pq, 6, 0);
    push(&pq, 7, 3);
    printf("Priority Queue elements:\n");
    display(pq);
    printf("\nElement with highest priority: %d\n", peek(&pq));
    pop(&pq);
    printf("\nPriority Queue after removing highest priority element:\n");
    display(pq);
    return 0;
}
```

```
/tmp/HkJlflIO73.o
Priority Queue elements:
Data: 6 Priority: 0
Data: 4 Priority: 1
Data: 5 Priority: 2
Data: 7 Priority: 3

Element with highest priority: 6

Priority Queue after removing highest priority element:
Data: 4 Priority: 1
Data: 5 Priority: 2
Data: 7 Priority: 3


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


```
/tmp/HkJlflIO73.o
Priority Queue elements:
Data: 6 Priority: 0
Data: 4 Priority: 1
Data: 5 Priority: 2
Data: 7 Priority: 3

Element with highest priority: 6

Priority Queue after removing highest priority element:
Data: 4 Priority: 1
Data: 5 Priority: 2
Data: 7 Priority: 3


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


2. write a c program for Binary Heap.

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_HEAP_SIZE 100
void heapify(int arr[], int n, int i);
void insert(int arr[], int* size, int key);
int extractMax(int arr[], int* size);
void display(int arr[], int size);
int main() {
    int heap[MAX_HEAP_SIZE];
    int size = 0;
    insert(heap, &size, 10);
    insert(heap, &size, 20);
    insert(heap, &size, 5);
    insert(heap, &size, 30);
    insert(heap, &size, 15);
    printf("Max-Heap elements:\n");
    display(heap, size);
    printf("\nExtracted max element: %d\n", extractMax(heap, &size));
    printf("\nMax-Heap elements after extraction:\n");
    display(heap, size);

    return 0;
}
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
```

```
/tmp/e6zDDjOOks.o
Max-Heap elements:
30 20 5 10 15

Extracted max element: 30

Max-Heap elements after extraction:
20 15 5 10


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


        arr[largest] = temp;


```c
        heapify(arr, n, largest);
    }
}
void insert(int arr[], int* size, int key) {
    if (*size >= MAX_HEAP_SIZE) {
        printf("Heap is full\n");
        return;
    }
    int i = *size;
    arr[i] = key;
    (*size)++;
    while (i != 0 && arr[(i - 1) / 2] < arr[i]) {
        int temp = arr[i];
        arr[i] = arr[(i - 1) / 2];
        arr[(i - 1) / 2] = temp;

        i = (i - 1) / 2;
    }
}
int extractMax(int arr[], int* size) {
    if (*size <= 0) return -1;
    if (*size == 1) {
        (*size)--;
        return arr[0];
    }
    int root = arr[0];
    arr[0] = arr[*size - 1];
    (*size)--;
    heapify(arr, *size, 0);

    return root;
}
void display(int arr[], int size) {
```


/tmp/e6zDDjOOks.o
Max-Heap elements:
30 20 5 10 15

Extracted max element: 30

Max-Heap elements after extraction:
20 15 5 10

=== Code Execution Successful ===

    return root;
}
void display(int arr[], int size) {


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

3. write a c program for Binary Search Tree.

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}
struct Node* search(struct Node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }
    if (data < root->data) {
        return search(root->left, data);
    } else {
        return search(root->right, data);
    }
}
```

```
/tmp/ZKd7UO15jN.o
Inorder Traversal of BST: 20 30 40 50 60 70 80
Node with value 40 found.
Inorder Traversal after deleting 20: 30 40 50 60 70 80


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

```c
}
struct Node* findMin(struct Node* root) {
    struct Node* current = root;
    while (current && current->left != NULL) {
        current = current->left;
    }
    return current;
}
struct Node* deleteNode(struct Node* root, int data) {
    if (root == NULL) {
        return root;
    }
    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
```


```
/tmp/ZKd7UO15jN.o
Inorder Traversal of BST: 20 30 40 50 60 70 80
Node with value 40 found.
Inorder Traversal after deleting 20: 30 40 50 60 70 80


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

```c
}
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);
    printf("Inorder Traversal of BST: ");
    inorderTraversal(root);
    printf("\n");
    int key = 40;
    struct Node* result = search(root, key);
    if (result != NULL) {
        printf("Node with value %d found.\n", key);
    } else {
        printf("Node with value %d not found.\n", key);
    }
    root = deleteNode(root, 20);
    printf("Inorder Traversal after deleting 20: ");
    inorderTraversal(root);
    printf("\n");
    return 0;
}
```

```
/tmp/ZKd7UO15jN.o
Inorder Traversal of BST: 20 30 40 50 60 70 80
Node with value 40 found.
Inorder Traversal after deleting 20: 30 40 50 60 70 80


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