1) Program:

```
#include <stdio.h>
#define MAX 100
int arr[MAX], n;
void display() {
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}
void insert(int pos, int val) {
   if (pos > n || pos < 0) {
      printf("Invalid position\n");
      return;
   }
   for (int i = n; i > pos; i--) {
      arr[i] = arr[i - 1];
   }
   arr[pos] = val;
   n++;
}
void delete(int pos) {
   if (pos >= n || pos < 0) {
      printf("Invalid position\n");
      return;
   }
   for (int i = pos; i < n - 1; i++) {
      arr[i] = arr[i + 1];
   }
   n--;   }
int search(int val) {
   for (int i = 0; i < n; i++) {
      if (arr[i] == val) {
         return i;
      }   }
   return -1;
}
int main() {
   n = 5;
   arr[0] = 10; arr[1] = 20; arr[2] = 30; arr[3] = 40; arr[4] = 50;
   printf("Array before operations: ");
   display();
   insert(2, 25);  // Insert 25 at position 2
   printf("Array after insertion: ");
   display();
   delete(3);  // Delete element at position 3
   printf("Array after deletion: ");
   display();
   int pos = search(25);  // Search for element 25
   if (pos != -1)
      printf("Element 25 found at position %d\n", pos);
   else
      printf("Element not found\n");
   return 0;
}
```

Output:

```
Array before operations: 10 20 30 40 50
Array after insertion: 10 20 25 30 40 50
Array after deletion: 10 20 25 40 50
Element 25 found at position 2
```

2) Program:

```
#include <stdio.h>
#define MAX 100
int stack[MAX], top = -1;
void push(int val) {
   if (top >= MAX - 1) {
      printf("Stack Overflow\n");
   } else {
      stack[++top] = val;
      printf("%d pushed to stack\n", val);
   }
}
void pop() {
   if (top < 0) {
      printf("Stack Underflow\n");
   } else {
      printf("%d popped from stack\n", stack[top--]);
   }
}
int search(int val) {
   for (int i = top; i >= 0; i--) {
      if (stack[i] == val) {
         return i;
      }
   }
   return -1;
}
void display() {
   if (top < 0) {
      printf("Stack is empty\n");
   } else {
      for (int i = top; i >= 0; i--) {
         printf("%d ", stack[i]);
      }
      printf("\n");
   }
}
int main() {
   push(10);
   push(20);
   push(30);
   display();
   pop();
   display();
   int pos = search(20);
   if (pos != -1)
      printf("Element found at position %d\n", pos);
   else
      printf("Element not found\n");
   return 0;
}
```

Output:

```
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 20 10
30 popped from stack
20 10
Element found at position 1
```

3) Program:

```
#include <stdio.h>
#define MAX 5
int queue[MAX], front = -1, rear = -1;
void enqueue(int val) {
   if (rear == MAX - 1) {
      printf("Queue Overflow\n");
   } else {
      if (front == -1) {
         front = 0;
      }
      queue[++rear] = val;
      printf("%d enqueued to queue\n", val);
   }   }
void dequeue() {
   if (front == -1 || front > rear) {
      printf("Queue Underflow\n");
   } else {
      printf("%d dequeued from queue\n", queue[front++]);
   }   }
int search(int val) {
   for (int i = front; i <= rear; i++) {
      if (queue[i] == val) {
         return i;
      }    }
   return -1;
}
void display() {
   if (front == -1 || front > rear) {
      printf("Queue is empty\n");
   } else {
      for (int i = front; i <= rear; i++) {
         printf("%d ", queue[i]);
      }
      printf("\n");
   }    }
int main() {
   enqueue(10);
   enqueue(20);
   enqueue(30);
   display();
   dequeue();
   display();
   int pos = search(20);
   if (pos != -1)
      printf("Element found at position %d\n", pos);
   else
      printf("Element not found\n");
   return 0;
}
```

Output:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
10 20 30
10 dequeued from queue
20 30
Element found at position 1
```

4) Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};
struct Node* head = NULL;
void insertAtBeginning(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = head;
   head = newNode;   }
void insertAtEnd(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = head;
   newNode->data = value;
   newNode->next = NULL;
   if (head == NULL) {
      head = newNode;
      return;
   }
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;   }
void insertAtPosition(int value, int position) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = head;
   newNode->data = value;
   if (position == 1) {
      newNode->next = head;
      head = newNode;
      return;    }
   for (int i = 1; i < position - 1 && temp != NULL; i++) {
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Position out of bounds.\n");
      return;  }
   newNode->next = temp->next;
   temp->next = newNode;   }
void display() {
   struct Node* temp = head;
   while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");     }
int main() {
   insertAtBeginning(10);
   insertAtEnd(20);
   insertAtEnd(30);
   insertAtPosition(15, 2);
   printf("Linked list after insertion operations: ");
   display();
   return 0;
}
```

Output:

Linked list after insertion operations: 10 -> 15 -> 20 -> 30 -> NULL

5) **Program:**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};
struct Node* head = NULL;
void insertAtBeginning(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = head;
   head = newNode;
}
void insertAtEnd(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = head;
   newNode->data = value;
   newNode->next = NULL;
   if (head == NULL) {
      head = newNode;
      return;
   }
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
}
void deleteAtBeginning() {
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }
   struct Node* temp = head;
   head = head->next;
   free(temp);
}
void deleteAtEnd() {
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }
   struct Node* temp = head;
   if (temp->next == NULL) {
      free(temp);
      head = NULL;
      return;
   }
   while (temp->next->next != NULL) {
      temp = temp->next;
   }
   free(temp->next);
   temp->next = NULL;
}
void deleteAtPosition(int position) {
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }
   struct Node* temp = head;
   if (position == 1) {
      head = temp->next;
      free(temp);
```

```
        return;
    }
    for (int i = 1; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("Position not found.\n");
        return;
    }
    struct Node* next = temp->next->next;
    free(temp->next);
    temp->next = next;
}
void display() {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    insertAtBeginning(10);
    insertAtEnd(20);
    insertAtEnd(30);
    insertAtPosition(15, 2);
    printf("Linked list after insertion operations: ");
    display();
    deleteAtBeginning();
    printf("After deletion at beginning: ");
    display();
    deleteAtEnd();
    printf("After deletion at end: ");
    display();
    deleteAtPosition(2);
    printf("After deletion at position 2: ");
    display();
    return 0;
}
```

Output:

```
Linked list after insertion operations: 10 -> 15 -> 20 -> 30 -> NULL
After deletion at beginning: 15 -> 20 -> 30 -> NULL
After deletion at end: 15 -> 20 -> NULL
After deletion at position 2: 15 -> NULL
```

6) Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};
struct Node* top = NULL;
void push(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = top;
   top = newNode;
}
void pop() {
   if (top == NULL) {
      printf("Stack Underflow\n");
      return;
   }
   struct Node* temp = top;
   top = top->next;
   free(temp);
}
int peek() {
   if (top != NULL)
      return top->data;
   else
      return -1;
}
void display() {
   struct Node* temp = top;
   while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}
int main() {
   push(10);
   push(20);
   push(30);
   printf("Stack: ");
   display();
   printf("Top element is %d\n", peek());
   pop();
   printf("Stack after pop: ");
   display();
   return 0;
}
```

Output:

```
Stack: 30 -> 20 -> 10 -> NULL
Top element is 30
Stack after pop: 20 -> 10 -> NULL
```

7) Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
   struct Node* prev;
};
struct Node* head = NULL;
void insertAtBeginning(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = head;
   newNode->prev = NULL;
   if (head != NULL)
      head->prev = newNode;
   head = newNode;
}
void insertAtEnd(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = head;
   newNode->data = value;
   newNode->next = NULL;
   if (head == NULL) {
      newNode->prev = NULL;
      head = newNode;
      return;
   }
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
   newNode->prev = temp;
}
void insertAtPosition(int value, int position) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = head;
   newNode->data = value;
   if (position == 1) {
      newNode->next = head;
      newNode->prev = NULL;
      if (head != NULL)
         head->prev = newNode;
      head = newNode;
      return;
   }
   for (int i = 1; temp != NULL && i < position - 1; i++) {
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Position out of bounds\n");
      return;
   }
   newNode->next = temp->next;
   newNode->prev = temp;
   if (temp->next != NULL)
      temp->next->prev = newNode;
   temp->next = newNode;
}
void display() {
   struct Node* temp = head;
   while (temp != NULL) {
      printf("%d <-> ", temp->data);
```

```
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    insertAtBeginning(10);
    insertAtEnd(20);
    insertAtEnd(30);
    insertAtPosition(15, 2);
    printf("Doubly Linked List: ");
    display();
    return 0;
}
```

Output:

Doubly Linked List: 10 <-> 15 <-> 20 <-> 30 <-> NULL

8) Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
   struct Node* prev;
};
struct Node* head = NULL;
void insertAtBeginning(int value) // Insertion at the beginning
 {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = head;
   newNode->prev = NULL;
   if (head != NULL) {
      head->prev = newNode;
   }
   head = newNode;
}
// Insertion at the end
void insertAtEnd(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = head;
   newNode->data = value;
   newNode->next = NULL;
   if (head == NULL) {
      newNode->prev = NULL;
      head = newNode;
      return;
   }
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
   newNode->prev = temp;
}
// Insertion at a specific position
void insertAtPosition(int value, int position) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node* temp = head;
   newNode->data = value;
   if (position == 1) {
      newNode->next = head;
      newNode->prev = NULL;
      if (head != NULL) {
         head->prev = newNode;
      }
      head = newNode;
      return;
   }
   for (int i = 1; temp != NULL && i < position - 1; i++) {
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Position out of bounds\n");
      return;
   }
   newNode->next = temp->next;
   newNode->prev = temp;
   if (temp->next != NULL) {
      temp->next->prev = newNode;
   }
```

```
    temp->next = newNode;
}
// Deletion at the beginning
void deleteAtBeginning() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    }
    free(temp);
}
// Deletion at the end
void deleteAtEnd() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    if (temp->next == NULL) {
        free(temp);
        head = NULL;
        return;
    }
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->prev->next = NULL;
    free(temp);
}
// Deletion at a specific position
void deleteAtPosition(int position) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    if (position == 1) {
        head = temp->next;
        if (head != NULL) {
            head->prev = NULL;
        }
        free(temp);
        return;
    }
    for (int i = 1; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("Position not found\n");
        return;
    }
    struct Node* next = temp->next->next;
    free(temp->next);
    temp->next = next;
    if (next != NULL) {
        next->prev = temp;
    }
}
// Displaying the list
```

```
void display() {
   struct Node* temp = head;
   while (temp != NULL) {
      printf("%d <-> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}
int main() {
   insertAtBeginning(10);
   insertAtEnd(20);
   insertAtEnd(30);
   insertAtPosition(15, 2);
   printf("Doubly Linked List: ");
   display();
   deleteAtBeginning();
   printf("After deletion at the beginning: ");
   display();
   deleteAtEnd();
   printf("After deletion at the end: ");
   display();
   deleteAtPosition(2);
   printf("After deletion at position 2: ");
   display();
   return 0;
}
```

Output:

```
Doubly Linked List: 10 <-> 15 <-> 20 <-> 30 <-> NULL
After deletion at the beginning: 15 <-> 20 <-> 30 <-> NULL
After deletion at the end: 15 <-> 20 <-> NULL
After deletion at position 2: 15 <-> NULL
```

9) Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};
// Create a new node
struct Node* newNode(int value) {
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->data = value;
   node->left = node->right = NULL;
   return node;
}
// Insert a new node in BST
struct Node* insert(struct Node* root, int value) {
   if (root == NULL) {
      return newNode(value);
   }
   if (value < root->data) {
      root->left = insert(root->left, value);
   } else {
      root->right = insert(root->right, value);
   }
   return root;
}
// Inorder traversal
void inorder(struct Node* root) {
   if (root != NULL) {
      inorder(root->left);
      printf("%d ", root->data);
      inorder(root->right);
   }
}
// Preorder traversal
void preorder(struct Node* root) {
   if (root != NULL) {
      printf("%d ", root->data);
      preorder(root->left);
      preorder(root->right);
   }
}
// Postorder traversal
void postorder(struct Node* root) {
   if (root != NULL) {
      postorder(root->left);
      postorder(root->right);
      printf("%d ", root->data);
   }
}
// Search a value in the BST
struct Node* search(struct Node* root, int value) {
   if (root == NULL || root->data == value) {
      return root;
   }
   if (value < root->data) {
      return search(root->left, value);
   } else {
      return search(root->right, value);
   }
}
int main() {
```

```
   struct Node* root = NULL;
   // Creating the BST
   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: ");
   inorder(root);
   printf("\n");
   printf("Preorder Traversal: ");
   preorder(root);
   printf("\n");
   printf("Postorder Traversal: ");
   postorder(root);
   printf("\n");
   int key = 40;
   struct Node* result = search(root, key);
   if (result != NULL) {
      printf("Node with value %d found in BST.\n", key); } else { printf("Node with value %d not found in BST.\n", key); }
return 0;
}
```

Output

```
Inorder Traversal: 20 30 40 50 60 70 80
Preorder Traversal: 50 30 20 40 70 60 80
 Postorder Traversal: 20 40 30 60 80 70 50
Node with value 40 found in BST.
```

10) Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};
// Create a new node
struct Node* newNode(int value) {
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->data = value;
   node->left = node->right = NULL;
   return node;
}
// Insert a new node in BST
struct Node* insert(struct Node* root, int value) {
   if (root == NULL) {
      return newNode(value);
   }
   if (value < root->data) {
      root->left = insert(root->left, value);
   } else {
      root->right = insert(root->right, value);
   }
   return root;
}
// Find the node with minimum value
struct Node* minValueNode(struct Node* node) {
   struct Node* current = node;
   while (current && current->left != NULL) {
      current = current->left;
   }
   return current;
}
// Delete a node from the BST
struct Node* deleteNode(struct Node* root, int value) {
   if (root == NULL) {
      return root;
   }
   if (value < root->data) {
      root->left = deleteNode(root->left, value);
   } else if (value > root->data) {
      root->right = deleteNode(root->right, value);
   } 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 = minValueNode(root->right);
      root->data = temp->data;
      root->right = deleteNode(root->right, temp->data);
   }
   return root;
}
// Inorder traversal
void inorder(struct Node* root) {
   if (root != NULL) {
```

```
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
// Search a value in the BST
struct Node* search(struct Node* root, int value) {
    if (root == NULL || root->data == value) {
        return root;
    }
    if (value < root->data) {
        return search(root->left, value);
    } else {
        return search(root->right, value);
    }
}
int main() {
    struct Node* root = NULL;
    // Creating the BST
    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: ");
    inorder(root);
    printf("\n");
    int key = 40;
    struct Node* result = search(root, key);
    if (result != NULL) {
        printf("Node with value %d found in BST.\n", key);
    } else {
        printf("Node with value %d not found in BST.\n", key);
    }
    root = deleteNode(root, 20);
    printf("Inorder Traversal after deleting 20: ");
    inorder(root);
    printf("\n");
    root = deleteNode(root, 30);
    printf("Inorder Traversal after deleting 30: ");
    inorder(root);
    printf("\n");
    return 0;
}
```

Output:

```
Inorder Traversal: 20 30 40 50 60 70 80
Node with value 40 found in BST.
Inorder Traversal after deleting 20: 30 40 50 60 70 80
Inorder Traversal after deleting 30: 40 50 60 70 80
```

11) Program:

```
#include <stdio.h>
// Bubble Sort
void bubbleSort(int arr[], int n) {
   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;
         }
      }
   }
}
// Insertion Sort
void insertionSort(int arr[], int n) {
   for (int i = 1; i < n; i++) {
      int key = arr[i];
      int j = i - 1;
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}
// Quick Sort
void quickSort(int arr[], int low, int high) {
   if (low < high) {
      int pivot = arr[high];
      int i = (low - 1);
      for (int j = low; j <= high - 1; j++) {
         if (arr[j] <= pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
         }
      }
      int temp = arr[i + 1];
      arr[i + 1] = arr[high];
      arr[high] = temp;

      int pi = i + 1;
      quickSort(arr, low, pi - 1);
      quickSort(arr, pi + 1, high);
   }
}
void printArray(int arr[], int n) {
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}
int main() {
   int arr[] = {64, 34, 25, 12, 22, 11, 90};
   int n = sizeof(arr) / sizeof(arr[0]);
   printf("Original array: ");
   printArray(arr, n);
   bubbleSort(arr, n);
   printf("Sorted array (Bubble Sort): ");
   printArray(arr, n);
   int arr2[] = {64, 34, 25, 12, 22, 11, 90};
```

```
    insertionSort(arr2, n);
    printf("Sorted array (Insertion Sort): ");
    printArray(arr2, n);
    int arr3[] = {64, 34, 25, 12, 22, 11, 90};
    quickSort(arr3, 0, n - 1);
    printf("Sorted array (Quick Sort): ");
    printArray(arr3, n);
    return 0;
}
```

Output:

```
Original array: 64 34 25 12 22 11 90
Sorted array (Bubble Sort): 11 12 22 25 34 64 90
Sorted array (Insertion Sort): 11 12 22 25 34 64 90
Sorted array (Quick Sort): 11 12 22 25 34 64 90
```

12) Program:

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10
struct Node {
   int key;
   int value;
   struct Node* next;
};
struct HashTable {
   struct Node* table[TABLE_SIZE];
};  // Initialize the hash table
void initTable(struct HashTable* ht) {
   for (int i = 0; i < TABLE_SIZE; i++) {
      ht->table[i] = NULL;
   }
}// Hash function
int hash(int key) {
   return key % TABLE_SIZE;
}// Insert a key-value pair into the hash table
void insert(struct HashTable* ht, int key, int value) {
   int index = hash(key);
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->key = key;
   newNode->value = value;
   newNode->next = ht->table[index];
   ht->table[index] = newNode;
}// Search for a value by key in the hash table
int search(struct HashTable* ht, int key) {
   int index = hash(key);
   struct Node* temp = ht->table[index];
   while (temp != NULL) {
      if (temp->key == key) {
         return temp->value;
      }
      temp = temp->next;
   }
   return -1; // Not found
}// Delete a key-value pair from the hash table
void delete(struct HashTable* ht, int key) {
   int index = hash(key);
   struct Node* temp = ht->table[index];
   struct Node* prev = NULL;
   while (temp != NULL && temp->key != key) {
      prev = temp;
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Key not found\n");
      return;
   }
   if (prev == NULL) {
      ht->table[index] = temp->next;
   } else {
      prev->next = temp->next;
   }
   free(temp);
}// Print the hash table
void printTable(struct HashTable* ht) {
   for (int i = 0; i < TABLE_SIZE; i++) {
      struct Node* temp = ht->table[i];
      printf("Index %d: ", i);
      while (temp != NULL) {
```

```
            printf("(%d, %d) -> ", temp->key, temp->value);
            temp = temp->next;
        }
        printf("NULL\n");
    } }
int main() {
    struct HashTable ht;
    initTable(&ht);
    insert(&ht, 1, 100);
    insert(&ht, 2, 200);
    insert(&ht, 12, 300);
    printf("Hash Table:\n");
    printTable(&ht);
    printf("Search for key 2: %d\n", search(&ht, 2));
    delete(&ht, 2);
    printf("After deleting key 2:\n");
    printTable(&ht);
    return 0;
}
```

Output:

```
Hash Table:
Index 0: (10, 300) -> NULL
Index 1: (1, 100) -> NULL
Index 2: (2, 200) -> NULL
Search for key 2: 200
After deleting key 2:
Index 0: (10, 300) -> NULL
Index 1: (1, 100) -> NULL
```