Write Linked List code with functions perform on it.
Code:

```
#include <iostream>

using namespace std;

class node {
public:
   int data;
   node* next;

   node(int val);
   node();
   void insertLast(int val);
   void insertFirst(int val);
   void insertAt(int val, int pos);
   void removeLast();
   void removeFirst();
   void removeAt(int pos);
   void display();
   void reverse();
   void sort();
   void search(int val);
};

node* head;
int node_count = 0;

node::node() {
   data = 0;
   next = NULL;
}

node::node(int val) {
   data = val;
   next = NULL;
}

void node::display() {
   node* trav = head;

   if (trav == NULL) {
      cout << "Stack is Empty" << endl;
   }

         else {
      cout << "\nData: ";
      while (trav != NULL) {
         cout << trav->data << " ";
         trav = trav->next;
      }
      cout << "\nTotal number of elements are " << node_count+1 << endl;
   }
}

void node::insertFirst(int val) {
   node* temp = new node(val);
   temp->next = head;
   head = temp;
   node_count++;
}

void node::removeFirst() {
   if (head == NULL) {
      cout << "Stack is Empty" << endl;
   } else {
      node* temp = head;
```

```
        head = head->next;
        delete temp;
        node_count--;
    }
}

void node::insertLast(int val) {
    node* temp = new node(val);
    if (head == NULL) {
        head = temp;
    } else {
        node* trav = head;
        while (trav->next != NULL) {
            trav = trav->next;
        }
        trav->next = temp;
    }
    node_count++;
}

void node::removeLast() {
    if (head == NULL) {
        cout << "Stack is Empty" << endl;
    } else if (node_count == 1) {
        delete head;
        head = NULL;
        node_count--;
    } else {
        node* trav = head;
        while (trav->next->next != NULL) {
            trav = trav->next;
        }
        delete trav->next;
        trav->next = NULL;
        node_count--;
    }
}

void node::insertAt(int val, int pos) {
    if (pos < 1 || pos > node_count + 1) {
        cout << "Wrong Choice" << endl;
    } else if (pos == 1) {
        insertFirst(val);
    } else if (pos == node_count + 1) {
        insertLast(val);
    } else {
        node* temp = new node(val);
        node* trav = head;
        int k = 1;
        while (k < pos - 1) {
            trav = trav->next;
            k++;
        }
        temp->next = trav->next;
        trav->next = temp;
        node_count++;
    }
}

void node::removeAt(int pos) {
    if (pos < 1 || pos > node_count) {
        cout << "Wrong Position" << endl;
    } else if (pos == 1) {
        removeFirst();
    } else if (pos == node_count) {
        removeLast();
    } else {
```

```
        node* trav = head;
        int k = 1;
        while (k < pos - 1) {
            trav = trav->next;
            k++;
        }
        node* temp = trav->next;
        trav->next = temp->next;
        delete temp;
        node_count--;
    }
}

void node::sort() {
    node* sort_Head = head;
    while (sort_Head != NULL) {
        node* trav = sort_Head->next;
        while (trav != NULL) {
            if (trav->data < sort_Head->data) {
                int temp = trav->data;
                trav->data = sort_Head->data;
                sort_Head->data = temp;
            }
            trav = trav->next;
        }
        sort_Head = sort_Head->next;
    }
}

void node::reverse() {
    if (head == NULL || head->next == NULL) {
        return; // Nothing to reverse
    }

    node* prev = NULL;
    node* current = head;
    node* nextNode;

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

    head = prev;
}

//searching
void node::search(int data){
            node* trav = head;
            bool flag = false;
            while(trav != NULL && flag == false){
                        if(trav -> data == data){
                                    flag = true;
                                    break;
                        }
                        else{
                                    trav = trav -> next;
                        }
            }
            if(flag == true){
                        cout << "Element Found!";
            }
            else{
                        cout << "Element Not Found!";
            }
```

```
}

int main() {
   head = new node(500);
            cout << "Insert Last";
            head->display();

   head->insertLast(200);
   cout << "Insert Last";
   head->display();

   head->insertLast(300);
   cout << "Insert Last";
   head->display();

   head->insertLast(400);
   cout << "Insert Last";
   head->display();

   head->insertFirst(700);
   cout << "Insert First";
   head->display();

   head->removeFirst();
   cout << "Remove First";
   head->display();

   head->insertAt(600, 1);
   cout << "Insert at First";
   head->display();

   head->insertAt(100, 3);
   cout << "Insert at First";
   head->display();

   head->removeFirst();
   cout << "Remove Last";
   head->display();

   head->insertLast(500);
   cout << "Insert Last";
   head->display();

   head->removeAt(4);
   cout << "Remove at Position 4";
   head->display();

   head->reverse();
   cout << "Reverse";
   head->display();

   head->sort();
   cout << "Sorted";
   head->display();

            cout<<"\nSearching Element in the Linked list: ";
            int n;
            cin >> n;
            head->search(n);
   return 0;
}
```

Output:

```
Insert Last
Data: 500
Total number of elements are 1
Insert Last
Data: 500 200
Total number of elements are 2
Insert Last
Data: 500 200 300
Total number of elements are 3
Insert Last
Data: 500 200 300 400
Total number of elements are 4
Insert First
Data: 700 500 200 300 400
Total number of elements are 5
Remove First
Data: 500 200 300 400
Total number of elements are 4
Insert at First
Data: 600 500 200 300 400
Total number of elements are 5
Insert at First
Data: 600 500 100 200 300 400
Total number of elements are 6
Remove Last
Data: 500 100 200 300 400
Total number of elements are 5
Insert Last
Data: 500 100 200 300 400 500
Total number of elements are 6
Remove at Position 4
Data: 500 100 200 400 500
Total number of elements are 5
Reverse
Data: 500 400 200 100 500
Total number of elements are 5
Sorted
Data: 100 200 400 500 500
Total number of elements are 5
```

```
Searching Element in the Linked list: 45
Element Not Found!
--------------------------------
Process exited after 188 seconds with return value 0
Press any key to continue . . .
```