```
#include<bits/stdc++.h>

using namespace std;

class Node
{
public:
    int value;
    Node* next;

    Node(int val)
    {
        value= val;
        next = NULL;
    }
};

struct Test
{
    int position[1000];
};

void insert_At_Head(Node* &head, int val);
void insert_At_Tail(Node* &head, int val);
void insert_At_Specific_Position(Node* &head,int pos, int val);

int search_by_unique_value(Node* head,int key);
void search_by_duplicate_value(Node* head,int key);
Test search_by_duplicate_value_return(Node* head,int key);
void search_by_value_unique(Node* &head,int searchValue, int val);

int count_of_length(Node* head);

void deletion_at_head(Node* &head);
void deletion_at_tail(Node* &head);
void deletion_At_Specific_Position(Node* &head,int val);
void deletion_by_value_unique(Node* &head,int val);

void display(Node* n);

void insert_At_Head(Node* &head, int val)
{
    Node *newNode = new Node(val);
    newNode->next = head;
    head = newNode;
}


void insert_At_Tail(Node* &head, int val)
{
    Node *newNode = new Node(val);

    if(head==NULL)
    {
        head = newNode;
        return;
    }

    Node *temp = head;

    while(temp->next!=NULL)
    {
        temp = temp->next;
    }

    temp->next = newNode;
}

void insert_At_Specific_Position(Node* &head,int pos, int val)
{
    int i =0;
    Node* temp = head;
    while(i<pos-2)
    {
        i++;
        temp = temp->next;
    }
    Node *newNode = new Node(val);
    newNode->next = temp->next;
    temp->next = newNode;
}

int search_by_unique_value(Node* head,int key)
{
    int count = 1;
    Node* temp = head;
    if(temp == NULL)
    {
        return -1;
    }
    while(temp->value != key)
    {
        if(temp->next == NULL)
        {
            return -1;
        }
        count++;
        temp = temp->next;
    }
    return count;
}

Test search_by_duplicate_value_return(Node* head,int key)
{
    Node* temp = head;
    Test T;
    int k=1;
    int count = 1;
    while(temp!= NULL)
    {
        if(temp->value == key)
        {
            //cout<<count<<" ";
            T.position[k]=count;
            k++;
        }
        temp = temp->next;
        count++;
    }
    T.position[0] = k;
    return T;

}

int count_of_length(Node* head)
{
    int count = 0;
    Node* temp = head;
    while(temp!=NULL)
    {
        count++;
        temp = temp->next;
    }
    return count;
}

void search_by_value_unique(Node* &head,int searchValue, int val)
{
    int position;
    position = search_by_unique_value(head,searchValue);
    insert_At_Specific_Position(head,position+1,val);
}

void deletion_at_head(Node* &head)
{
    Node* temp = head;
    if(temp!=NULL)
    {
        head = temp->next;
        delete temp;
    }

    else
    {
        cout<<"There is no value in the linked list"<<endl;
    }
}

void deletion_at_tail(Node* &head)
{
    Node* temp = head;
    if(temp!=NULL && temp->next!=NULL)
    {
        while(temp->next->next!=NULL)
        {
            temp = temp->next;
        }
        Node* delNode = temp->next;
        temp->next = NULL;
        delete delNode;
    }
    else
    {
        if(temp == NULL)
        {
            cout<<"There is no value in the linked list"<<endl;
        }
        else
        {
            deletion_at_head(head);
        }
    }
}

void deletion_At_Specific_Position(Node* &head,int pos)
{
    Node* temp = head;
    if(pos <= count_of_length(head))
    {
        if(pos==1)
        {
            deletion_at_head(head);
        }
        else if (pos == count_of_length(head))
        {
            deletion_at_tail(head);
        }
        else
        {
            int i = 1;
            while(i<pos-1)
            {
                temp = temp->next;
                i++;
            }

            Node* delNode = temp->next;
            temp->next = delNode->next;
            delete delNode;
        }

    }
    else
    {
        cout<<"Position out of Bound";
    }
}

void deletion_by_value_unique(Node* &head,int val)
```

```
{
    int position;
    position = search_by_unique_value(head,val);
    if(position == -1)
    {
        cout<<"There is no value in the linked list"<<endl;
    }
    else
    {
        deletion_At_Specific_Position(head,position);
    }
}

int find_mid(Node* &head)
{
    Node* fast = head;
    Node* slow = head;
    if(head == NULL)
    {
        return -1;
    }

    while(fast != NULL && fast->next != NULL)
    {
        fast = fast->next->next;
        slow = slow->next;
    }
    return slow->value;
}

void make_cycle(Node* &head, int pos)
{
    Node* temp = head;
    Node* startNode = head;
    int count = 1;

    while (temp->next != NULL)
    {
        if(count == pos ) startNode = temp;
        temp = temp->next;
        count++;
    }

    temp->next = startNode;
```

```
}

bool detect_cycle(Node* &head)
{
    Node* fast = head;
    Node* slow = head;
    while(fast != NULL && fast->next != NULL)
    {
        fast = fast->next->next;
        slow = slow->next;
        if(slow->next == fast->next)
        {
            return true;
        }
    }
    return false;

}

void remove_cycle(Node* &head)
{
    Node* fast = head;
    Node* slow = head;

    do
    {
        fast = fast->next->next;
        slow = slow->next;
    }while(slow != fast);

    fast = head;

    while(fast->next != slow->next)
    {
        slow = slow->next;
        fast = fast->next;
    }
    slow->next = NULL;
}

void display(Node* n)
{
    while(n!=NULL)
    {
        cout<< n->value;
        if(n->next!=NULL) cout<<" -> ";
        n = n->next;
    }

    cout<<endl<<endl;

}

int main()
{
    Node *head = NULL;

    int n,pos;

    cout<<"Choice 1: Insertion at Head" << endl
        <<"Choice 2: Insertion at Tail" << endl
        <<"Choice 3: Insertion at Specific Position" << endl
        <<"Choice 4: Search a Value (Unique List)" << endl
        <<"Choice 5: Search a Value (Duplication Enabled List)" << endl
        <<"Choice 6: Insertion after a Specfic Value (Unique List)" << endl
        <<"Choice 7: Deletion at Head" << endl
        <<"Choice 8: Deletion at Tail" << endl
        <<"Choice 9: Deletion at Specific Position" << endl
        <<"Choice 10: Deletion by Value (Unique List)" << endl
        <<"Choice 11: Find mid point : " << endl
        <<"Choice 12: Make a cycle : " << endl
        <<"Choice 13: Detect a cycle : " << endl
        <<"Choice 14: Remove the cycle (if any): " << endl
        <<"Choice 0 : Exit" << endl
        <<"---------------------------------------------------" << endl <<endl;
    cout<<"Choice :";
    int choice ;
    cin>>choice;

    while(choice != 0)
    {
        switch(choice)
        {
        case 1:
            cout<<"Enter The Value : ";
            cin>>n;
            insert_At_Head(head,n);
            break;
```

```
case 2:
    cout<<"Enter The Value : ";
    cin>>n;
    insert_At_Tail(head,n);
    break;

case 3:
    cout<<"Enter The Position : ";
    cin>>pos;
    cout<<"Enter The Value : ";
    cin>>n;
    insert_At_Specific_Position(head,pos,n);
    break;
case 4:
    cout<<"Enter The Value to Search : ";
    cin>>n;
    pos = search_by_unique_value(head,n);
    if(pos != 1)
    {
        cout<<"The number is at position : "<<pos<<endl;
    }
    else
    {
        cout<<"The number is not yet in the list"<<endl;
    }
    break;

case 5:
    cout<<"Enter The Value to Search : ";
    cin>>n;

    Test T;
    T = search_by_duplicate_value_return(head,n);

    if(T.position[0] == 1)
    {
        cout<<"The number is not yet in the list"<<endl;
    }
    else
    {
        int size = T.position[0];
        cout<<"The value is found at position : ";
        for(int i=1; i<size; i++)
        {
            cout<<T.position[i];
            if(i<size-1)
            {
                cout<<" , ";
            }
        }
        cout<<endl;
    }
    break;

case 6:
    cout<<"Enter The Value to Search : ";
    int searchValue;
    cin>>searchValue;
    cout<<"Enter The Value to Insert : ";
    cin>>n;

    search_by_value_unique(head,searchValue,n);
    break;

case 7:
    deletion_at_head(head);
    break;

case 8:
    deletion_at_tail(head);
    break;
case 9:
    if(head == NULL)
    {
        cout<<"There is no value in the linked list"<<endl;
        break;
    }
    cout<<"Enter The Position : ";
    cin>>pos;
    deletion_At_Specific_Position(head,pos);
    break;
case 10:
    cout<<"Enter The Value to Delete : ";
    int delValue;
    cin>>delValue;
    deletion_by_value_unique(head,delValue);
    break;
    case 11:
        int mid;
        if(mid == -1) cout<<"The Linked list is empty."<<endl;
        mid = find_mid(head);
        cout<<"The mid point is : "<<mid<<endl;

    case 12:
        cout<<"Enter The Position : ";
        cin>>pos;
        make_cycle(head,pos);
        break;

    case 13:
        bool cycle_status;
        cycle_status = detect_cycle(head);
        if(cycle_status == true)
        {
            cout<<"There is a cycle in the list."<<endl;
        }
        else
        {
          cout<<"There is no cycle in the list."<<endl;
        }
        break;

    case 14:
        cycle_status = detect_cycle(head);
        if(cycle_status == true)
        {
            remove_cycle(head);
        }
        else
        {
          cout<<"There is no cycle in the list."<<endl;
        }
        break;

    default:
        break;
    }
    cout<<"Next Choice :";
    cin>>choice;
}
```

```
    cout<<endl;
    cout<<"Linked List : ";
    display(head);
    return 0;
}
```