Q 11. write a program in c++ to perform insertion and deletion operation in queue using dynamic implementation

```
#include <iostream>
struct node {
   int data;
   struct node *next;
};
struct node* front = NULL;
struct node* rear = NULL;
struct node* temp;
void Insert() {
   int val;
   cout<<"Insert the element in queue : "<<endl;
   cin>>val;
   if (rear == NULL) {
      rear = (struct node *)malloc(sizeof(struct node));
      rear->next = NULL;
      rear->data = val;
      front = rear;
   } else {
      temp=(struct node *)malloc(sizeof(struct node));
      rear->next = temp;
      temp->data = val;
      temp->next = NULL;
      rear = temp;
   }
}
void Delete() {
   temp = front;
   if (front == NULL) {
      cout<<"Underflow"<<endl;
      return;
   }
   else
   if (temp->next != NULL) {
      temp = temp->next;
      cout<<"Element deleted from queue is : "<<front-
>data<<endl;
      free(front);
      front = temp;
```

```
   } else {
      cout<<"Element deleted from queue is : "<<front-
>data<<endl;
      free(front);
      front = NULL;
      rear = NULL;
   }
}
void Display() {
   temp = front;
   if ((front == NULL) && (rear == NULL)) {
      cout<<"Queue is empty"<<endl;
      return;
   }
   cout<<"Queue elements are: ";
   while (temp != NULL) {
      cout<<temp->data<<" ";
      temp = temp->next;
   }
   cout<<endl;
}
int main() {
   int ch;
   cout<<"1) Insert element to queue"<<endl;
   cout<<"2) Delete element from queue"<<endl;
   cout<<"3) Display all the elements of queue"<<endl;
   cout<<"4) Exit"<<endl;
do {
   cout<<"Enter your choice : "<<endl;
   cin>>ch;
   switch (ch) {
      case 1: Insert();
         break;
      case 2: Delete();
         break;
      case 3: Display();
         break;
      case 4: cout<<"Exit"<<endl;
         break;
      default: cout<<"Invalid choice"<<endl;
   }
```

```
} while(ch!=4);
   return 0;
}
```

The output of the above program is as follows

```
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit
Enter your choice : 1
Insert the element in queue : 4
Enter your choice : 1
Insert the element in queue : 3
Enter your choice : 1
Insert the element in queue : 5
Enter your choice : 2
Element deleted from queue is : 4
Enter your choice : 3
Queue elements are : 3 5
Enter your choice : 7
Invalid choice
Enter your choice : 4
Exit
```

12. write a program in c++ insert a node at beginning in singly linked list

```
#include <iostream>

struct Node {

   int data;

   struct Node *next;

};

struct Node* head = NULL;

void insert(int new_data) {

   struct Node* new_node = (struct Node*)
malloc(sizeof(struct Node));

   new_node->data = new_data;

   new_node->next = head;

   head = new_node;

}

void display() {

   struct Node* ptr;

   ptr = head;

   while (ptr != NULL) {

      cout<< ptr->data <<" ";

      ptr = ptr->next;
```

```
    }
}
int main() {
    insert(3);
    insert(1);
    insert(7);
    insert(2);
    insert(9);
    cout<<"The linked list is: ";
    display();
    return 0;
}
```

Output

```
The linked list is: 9 2 7 1 3
```

13. write a program in c++ insert a node at middle in singly linked list

```
#include <bits/stdc++.h>
// structure of a node
struct Node {
    int data;
    Node* next;
};

// function to create and return a node
Node* getNode(int data)
{
    // allocating space
    Node* newNode = (Node*)malloc(sizeof(Node));

    // inserting the required data
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// function to insert node at the middle
// of the linked list
void insertAtMid(Node** head_ref, int x)
{
    // if list is empty
    if (*head_ref == NULL)
        *head_ref = getNode(x);
    else {

        // get a new node
        Node* newNode = getNode(x);

        Node* ptr = *head_ref;
        int len = 0;

        // calculate length of the linked list
        //, i.e, the number of nodes
        while (ptr != NULL) {
            len++;
            ptr = ptr->next;
        }

        // 'count' the number of nodes after which
        //  the new node is to be inserted
        int count = ((len % 2) == 0) ? (len / 2) :
                                    (len + 1) / 2;
        ptr = *head_ref;

        // 'ptr' points to the node after which
        // the new node is to be inserted
        while (count-- > 1)
```

```
            ptr = ptr->next;

        // insert the 'newNode' and adjust the
        // required links
        newNode->next = ptr->next;
        ptr->next = newNode;
    }
}

// function to display the linked list
void display(Node* head)
{
    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
    }
}

// Driver program to test above
int main()
{
    // Creating the list 1->2->4->5
    Node* head = NULL;
    head = getNode(1);
    head->next = getNode(2);
    head->next->next = getNode(4);
    head->next->next->next = getNode(5);

    cout << "Linked list before insertion: ";
    display(head);

    int x = 3;
    insertAtMid(&head, x);

    cout << "\nLinked list after insertion: ";
    display(head);

    return 0;
}
```

```
Linked list before insertion: 1 2 4 5

Linked list after insertion: 1 2 3 4 5
```

14. write a program in c++ insert a node at last in singly linked list

```
// A linked list Node
struct Node {
    int data;
    struct Node* next;
};

// Size of linked list
int size = 0;

// function to create and return a Node
Node* getNode(int data)
{
    // allocating space
    Node* newNode = new Node();

    // inserting the required data
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// function to insert a Node at required postion
void insertPos(Node** current, int pos, int data)
{
    // This condition to check whether the
    // postion given is valid or not.
    if (pos < 1 || pos > size + 1)
        cout << "Invalid postion!" << endl;
    else {

        // Keep looping until the pos is zero
        while (pos--) {

            if (pos == 0) {

                // adding Node at required postion
                Node* temp = getNode(data);

                // Making the new Node to point to
                // the old Node at the same position
                temp->next = *current;

                // Changing the pointer of the Node previous
                // to the old Node to point to the new Node
                *current = temp;
            }
            else
              // Assign double pointer variable to point to the
```

```
            // pointer pointing to the address of next Node
            current = &(*current)->next;
        }
        size++;
    }
}

// This function prints contents
// of the linked list
void printList(struct Node* head)
{
    while (head != NULL) {
        cout << " " << head->data;
        head = head->next;
    }
    cout << endl;
}

// Driver Code
int main()
{
    // Creating the list 3->5->8->10
    Node* head = NULL;
    head = getNode(3);
    head->next = getNode(5);
    head->next->next = getNode(8);
    head->next->next->next = getNode(10);

    size = 4;

    cout << "Linked list before insertion: ";
    printList(head);

    int data = 12, pos = 3;
    insertPos(&head, pos, data);
    cout << "Linked list after insertion of 12 at position 3: ";
    printList(head);

    // front of the linked list
    data = 1, pos = 1;
    insertPos(&head, pos, data);
    cout << "Linked list after insertion of 1 at position 1: ";
    printList(head);

    // insetion at end of the linked list
    data = 15, pos = 7;
    insertPos(&head, pos, data);
    cout << "Linked list after insertion of 15 at position 7: ";
    printList(head);

    return 0;
}
```

Output:

```
Linked list before insertion:  3 5 8 10
Linked list after insertion of 12 at position 3:  3 5 12 8 10
Linked list after insertion of 1 at position 1:  1 3 5 12 8 10
Linked list after insertion of 15 at position 7:  1 3 5 12 8 10 15
```

15. write a program in c++ to delete a node at begining in singly linked list.

```
/* Structure of a node */
struct node {
    int data;           // Data
    struct node *next; // Address
}*head;


void createList(int n);
void deleteFirstNode();
void displayList();



int main()
{
    int n, choice;

    /*
     * Create a singly linked list of n nodes
     */
    cout<<”Enter the total number of nodes: ";
    cin>>&n;
    createList(n);

    cout<<“Data in the list \n;
    displayList();

    cout<<"\nPress 1 to delete first node: ";
    cin>>choice;

    /* Delete first node from list */
    if(choice == 1)
        deleteFirstNode();

    cout<<"\nData in the list \n";
    displayList();

    return 0;
}
```

```
void createList(int n)
{
    struct node *newNode, *temp;
    int data, i;

    head = (struct node *)malloc(sizeof(struct node));

    /*
     * If unable to allocate memory for head node
     */
    if(head == NULL)
    {
        cout<<"Unable to allocate memory.";
    }
    else
    {
        /*
         * In data of node from the user
         */
         cout<<"Enter the data of node 1: ";
        cin>>data;

        head->data = data; // Link the data field with data
        head->next = NULL; // Link the address field to NULL

        temp = head;

        /*
         * Create n nodes and adds to linked list
         */
        for(i=2; i<=n; i++)
        {
            newNode = (struct node *)malloc(sizeof(struct node));

            /* If memory is not allocated for newNode */
            if(newNode == NULL)
            {
                cout<<"Unable to allocate memory.";
                break;
            }
            else
            {
                cout<<"Enter the data of node %d: "<< i;
                cin>>data;

                newNode->data = data; // Link the data field of newNode
with data
```

```
                newNode->next = NULL; // Link the address field of newNode
with NULL

                temp->next = newNode; // Link previous node i.e. temp to
the newNode
                temp = temp->next;
            }
        }

        Cout<<"SINGLY LINKED LIST CREATED SUCCESSFULLY\n";
    }
}


void deleteFirstNode()
{
    struct node *toDelete;

    if(head == NULL)
    {
        cout<<"List is already empty.";
    }
    else
    {
        toDelete = head;
        head = head->next;

        cout<<"\nData deleted" <<toDelete->data;

        /* Clears the memory occupied by first node*/
        free(toDelete);

        cout<<"SUCCESSFULLY DELETED FIRST NODE FROM LIST\n";
    }
}
```

Output ----------

```
of nodes: 5
Enter the data of node 1: 10
Enter the data of node 2: 20
Enter the data of node 3: 30
Enter the data of node 4: 40
```

```
Enter the data of node 5: 50
SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40
Data = 50

Press 1 to delete first node: 1

Data deleted = 10
SUCCESSFULLY DELETED FIRST NODE FROM LIST

Data in the list
Data = 20
Data = 30
Data = 40
Data = 50
```

16. write a program in c++ to delete a node at last in singly linked list.

```
/* Structure of a node */
struct node {
    int data;           // Data
    struct node *next; // Address
}*head;


void createList(int n);
void deleteFirstNode();
void displayList();



int main()
{
    int n, choice;

    /*
     * Create a singly linked list of n nodes
     */
    cout<<”Enter the total number of nodes: ";
    cin>>&n;
    createList(n);

    cout<<“Data in the list \n;
    displayList();

    cout<<"\nPress 1 to delete first node: ";
    cin>>choice;

    /* Delete first node from list */
    if(choice == 1)
        deleteLastNode();

    cout<<"\nData in the list \n";
    displayList();

    return 0;
}
```

```
void createList(int n)
{
    struct node *newNode, *temp;
    int data, i;

    head = (struct node *)malloc(sizeof(struct node));

    /*
     * If unable to allocate memory for head node
     */
    if(head == NULL)
    {
        cout<<"Unable to allocate memory.";
    }
    else
    {
        /*
         * In data of node from the user
         */
         cout<<"Enter the data of node 1: ";
        cin>>data;

        head->data = data; // Link the data field with data
        head->next = NULL; // Link the address field to NULL

        temp = head;

        /*
         * Create n nodes and adds to linked list
         */
        for(i=2; i<=n; i++)
        {
            newNode = (struct node *)malloc(sizeof(struct node));

            /* If memory is not allocated for newNode */
            if(newNode == NULL)
            {
                cout<<"Unable to allocate memory.";
                break;
            }
            else
            {
                cout<<"Enter the data of node %d: "<< i;
                cin>>data;

                newNode->data = data; // Link the data field of newNode
with data
```

```
                newNode->next = NULL; // Link the address field of newNode
with NULL

                temp->next = newNode; // Link previous node i.e. temp to
the newNode
                temp = temp->next;
            }
        }

        Cout<<"SINGLY LINKED LIST CREATED SUCCESSFULLY\n";
    }
}


void deleteLastNode()
{
    struct node *toDelete;

    if(head == NULL)
    {
        cout<<"List is already empty.";
    }
    else
    {
        toDelete = head;
        secondLastNode = head;

        /* Traverse to the last node of the list */
        while(toDelete->next != NULL)
        {
            secondLastNode = toDelete;
            toDelete = toDelete->next;
        }

        if(toDelete == head)
        {
            head = NULL;
        }
        else
        {
            /* Disconnect link of second last node with last node */
            secondLastNode->next = NULL;
        }

        /* Delete the last node */
        free(toDelete);
```

```
        cout<<"SUCCESSFULLY DELETED LAST NODE OF LIST\n";
    }
}
```

Out Put

```
Enter the total number of nodes: 5
Enter the data of node 1: 10
Enter the data of node 2: 20
Enter the data of node 3: 30
Enter the data of node 4: 40
Enter the data of node 5: 50
SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40
Data = 50

Press 1 to delete last node: 1
SUCCESSFULLY DELETED LAST NODE OF LIST

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40
```

17. write a program in c++ to delete a node at middle in singly linked list.

```
/* Structure of a node */
struct node {
    int data;           // Data
    struct node *next; // Address
}*head;


void createList(int n);
void deleteFirstNode();
void displayList();



int main()
{
    int n, choice;

    /*
     * Create a singly linked list of n nodes
     */
    cout<<”Enter the total number of nodes: ";
    cin>>&n;
    createList(n);

    cout<<“Data in the list \n;
    displayList();

    cout<<"\nPress 1 to delete first node: ";
    cin>>choice;

    /* Delete first node from list */
    if(choice == 1)
        deleteMiddleNode();

    cout<<"\nData in the list \n";
    displayList();

    return 0;
}
```

```
void createList(int n)
{
    struct node *newNode, *temp;
    int data, i;

    head = (struct node *)malloc(sizeof(struct node));

    /*
     * If unable to allocate memory for head node
     */
    if(head == NULL)
    {
        cout<<"Unable to allocate memory.";
    }
    else
    {
        /*
         * In data of node from the user
         */
         cout<<"Enter the data of node 1: ";
        cin<<data;

        head->data = data; // Link the data field with data
        head->next = NULL; // Link the address field to NULL

        temp = head;

        /*
         * Create n nodes and adds to linked list
         */
        for(i=2; i<=n; i++)
        {
            newNode = (struct node *)malloc(sizeof(struct node));

            /* If memory is not allocated for newNode */
            if(newNode == NULL)
            {
                cout<<"Unable to allocate memory.";
                break;
            }
            else
            {
                cout<<"Enter the data of node %d: "<< i;
                cin>>data;

                newNode->data = data; // Link the data field of newNode
with data
```

```
                newNode->next = NULL; // Link the address field of newNode
with NULL

                temp->next = newNode; // Link previous node i.e. temp to
the newNode
                temp = temp->next;
            }
         }

         cout<<"SINGLY LINKED LIST CREATED SUCCESSFULLY\n";
    }
}


void deleteMiddleNode()
{
    struct node *toDelete;

    if(head == NULL)
    {
        cout<<"List is already empty.";
    }
    else
    {
      toDelete = head;
        prevNode = head;

        for(i=2; i<=position; i++)
        {
            prevNode = toDelete;
            toDelete = toDelete->next;

            if(toDelete == NULL)
                break;
        }

        if(toDelete != NULL)
        {
            if(toDelete == head)
                head = head->next;

            prevNode->next = toDelete->next;
            toDelete->next = NULL;

            /* Delete nth node */
            free(toDelete);

            cout<<"SUCCESSFULLY DELETED NODE FROM MIDDLE OF LIST\n";
        }
        else
```

```
        {
            cout<<"Invalid position unable to delete.";
        }
}
```

Out Put

```
Enter the total number of nodes: 5
Enter the data of node 1: 10
Enter the data of node 2: 20
Enter the data of node 3: 30
Enter the data of node 4: 40
Enter the data of node 5: 50
SINGLY LINKED LIST CREATED SUCCESSFULLY

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40
Data = 50

Enter the node position you want to delete: 4
SUCCESSFULLY DELETED NODE FROM MIDDLE OF LIST

Data in the list
Data = 10
Data = 20
Data = 30
Data = 50
```

18. wap Program to Implement Circular Singly Linked List

```
#include <iostream>
struct Node {
   int data;
   struct Node *next;
};
struct Node* head = NULL;
void insert(int newdata) {
   struct Node *newnode = (struct Node *)malloc(sizeof(struct Node));
   struct Node *ptr = head;
   newnode->data = newdata;
   newnode->next = head;
   if (head!= NULL) {
      while (ptr->next != head)
         ptr = ptr->next;
      ptr->next = newnode;
   } else
      newnode->next = newnode;
      head = newnode;
}
void display() {
   struct Node* ptr;
   ptr = head;
   do {
      cout<<ptr->data <<" ";
      ptr = ptr->next;
```

```
   } while(ptr != head);
}
int main() {
   insert(3);
   insert(1);
   insert(7);
   insert(2);
   insert(9);
   cout<<"The circular linked list is: ";
   display();
   return 0;
}
```

Output

```
The circular linked list is: 9 2 7 1 3
```

19. wap Program to perform all insertion operation in singly linked list.

```
#include<bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    struct Node *next;
};

struct Node *addToEmpty(struct Node *last, int data)
{
    // This function is only for empty list
    if (last != NULL)
      return last;

    // Creating a node dynamically.
    struct Node *temp =
           (struct Node*)malloc(sizeof(struct Node));

    // Assigning the data.
    temp -> data = data;
    last = temp;

    // Creating the link.
    last -> next = last;

    return last;
}

struct Node *addBegin(struct Node *last, int data)
{
    if (last == NULL)
        return addToEmpty(last, data);

    struct Node *temp =
            (struct Node *)malloc(sizeof(struct Node));

    temp -> data = data;
    temp -> next = last -> next;
    last -> next = temp;

    return last;
}

struct Node *addEnd(struct Node *last, int data)
{
    if (last == NULL)
        return addToEmpty(last, data);
```

```
    struct Node *temp =
        (struct Node *)malloc(sizeof(struct Node));

    temp -> data = data;
    temp -> next = last -> next;
    last -> next = temp;
    last = temp;

    return last;
}

struct Node *addAfter(struct Node *last, int data, int item)
{
    if (last == NULL)
        return NULL;

    struct Node *temp, *p;
    p = last -> next;
    do
    {
        if (p ->data == item)
        {
            temp = (struct Node *)malloc(sizeof(struct Node));
            temp -> data = data;
            temp -> next = p -> next;
            p -> next = temp;

            if (p == last)
                last = temp;
            return last;
        }
        p = p -> next;
    }  while(p != last -> next);

    cout << item << " not present in the list." << endl;
    return last;

}

void traverse(struct Node *last)
{
    struct Node *p;

    // If list is empty, return.
    if (last == NULL)
    {
        cout << "List is empty." << endl;
        return;
    }

    // Pointing to first Node of the list.
    p = last -> next;
```

```
    // Traversing the list.
    do
    {
        cout << p -> data << " ";
        p = p -> next;

    }
    while(p != last->next);

}

// Driven Program
int main()
{
    struct Node *last = NULL;

    last = addToEmpty(last, 6);
    last = addBegin(last, 4);
    last = addBegin(last, 2);
    last = addEnd(last, 8);
    last = addEnd(last, 12);
    last = addAfter(last, 10, 8);

    traverse(last);

    return 0;
}
```

Output:

```
2 4 6 8 10 12
```