DATA STRUCTURE USING C++

Name: BIND SUNNY SANTOSH
Roll No: CS22009

ATMANAND SARASWATI SCIENCE COLLEGE
KAPODRA, VARACHHA ROAD, SURAT – 395006.

1. Write Stack Code using all operation (Push, Pop, Peep).

INPUT:

```
#include<iostream>
#include<string>
using namespace std;

class stack{
    private:
    int top;
    int arr[5];
    public:
        stack()
        {     top=-1;
              for(int i=0;i<5;i++)
              {
               arr[i]=0;
               }
        }
        bool isEmpty()
        {   if(top==-1)
            {
                    return true;
            }
            else
            {
               return false;
            }
        }
```

```
 bool isfull()
{    if(top==4)
     {
               return true;
     }
     else
     {
        return false;
     }
}
void push(int value)
{
    if(isfull())
    {
         cout<<"Stack overflow"<<endl;
    }
    else
    {
      top++;
      arr[top]=value;
    }
}
int pop()
{
    if(isEmpty())
    {
       cout<<"stack underflow"<<endl;
    }
    else
    {   int popvalue=arr[top];
```

```
        arr[top]=0;
        top--;
        return popvalue;
    }
}
int count()
{
return (top+1);
}
int peek(int position)
{
    if(isEmpty())
    {
     cout<<"stack underflow"<<endl;
                          return -1;
    }
                         else if (position < 0 || position > top)
                         {
                        cout << "Invalid position" << endl;
                        return -1;
                                    }
    else
    {
        return arr[position];
    }
}
void change(int position,int value)
{
    arr[position]=value;
}
```

```
        void display()
        {
            for(int i=4;i>-1;i--)
            {
             cout<<arr[i]<<endl;
            }
        }
};

int main()
{
    stack s1;
    int op,position,value;
    do
    {
     cout<<"what operation do you want to perform? "<<endl;
     cout<<"0.exit"<<endl;
     cout<<"1.push"<<endl;
     cout<<"2.pop"<<endl;
     cout<<"3.isEmpty"<<endl;
     cout<<"4.isfull"<<endl;
     cout<<"5.peek"<<endl;
     cout<<"6.count"<<endl;
     cout<<"7.change"<<endl;
     cout<<"8.display"<<endl;
     cout<<"9.clear"<<endl;

     cout<<"Enter your choice ?"<<endl;
     cin>>op;
```

```
switch(op)
{

    case 0 :
          break;

    case 1 :
        cout<<"Enter the value to insert into the stack : "<<endl;
        cin>>value;
        s1.push(value);
        cout<<"Data Inserted in stack";
        break;

    case 2 :
        cout<<"pop function called : "<<s1.pop()<<endl;
        cout<<"Data deleted from stack"<<endl;
        break;

    case 3 :
        if(s1.isEmpty())
        {
            cout<<"Stack is Empty"<<endl;
        }
        else
        {
            cout<<"Stack is not Empty"<<endl;
        }
        break;

    case 4 :
        if(s1.isfull())
        {
            cout<<"Stack is Full"<<endl;
        }
        else
        {
            cout<<"Stack is not Full"<<endl;
        }

        break;

    case 5 :
        cout<<"Enter the position you want to peek into the stack : "<<endl;
        cin>>position;
        s1.peek(position);
        cout<<"The value is "<<s1.peek(position)<<endl;


        break;

    case 6 :
        cout<<"Total no. of item  into the stack is : "<<s1.count()<<endl;
        break;

    case 7 :
        cout<<"Enter the position the stack : "<<endl;
        cin>>position;
        cout<<"Enter the value to insert into the stack : "<<endl;
        cin>>value;
        s1.change(position,value);
        break;
```

```
                case 8 :
                    cout<<"The item of stacks are : "<<endl;
                    s1.display();
                    break;

                case 9 :
                    system("cls");
                    break;

                default:
                     cout<<"enter the correct value!!";
                     break;
          }
          }
          while(op!=0);
          return 0;
      }
```

OUTPUT:

```
what operation do you want to perform?
0.exit
1.push
2.pop
3.isEmpty
4.isfull
5.peek
6.count
7.change
8.display
9.clear
```

```
Enter your choice ?

8

The item of stacks are :

4

6

8

5

4
```

2. Write Queue Code using all operation (insert, delete and view).

INPUT:

```
#include<iostream>
#include<string>
using namespace std;

class queue
{
  private:
            int rear;
            int front;
            int arr[5];
            public:
            queue()
            {
                    rear=-1;
                    front=-1;
                    for(int i=0;i<5;i++)
                    {
```

```
                    arr[i]=0;
          }
}

bool isEmpty()
{
          if(rear==-1 && front==-1)
          {
                    return true;
          }
          else
          {
                    return false;
          }
}

bool isFull()
{
          if(rear==4)
          {
                    return true;
          }
          else
          {
                    return false;
          }
}

void enqueue(int value)
{
        if(isFull())
        {
                cout<<"Queue is Full"<<endl;
                return;
        }
        else if(isEmpty())
        {
                rear=0;
                front=0;
                arr[rear]=value;
        }
        else
        {
                rear++;
                arr[rear]=value;
        }
}

int dequeue()
{
        int x;
        if(isEmpty())
        {
                cout<<"Queue is empty"<<endl;
                return 0;
        }
        else if(rear==front)
        {
                x=arr[front];
                arr[front]=0;
```

```
                        front=-1;
                        rear=-1;
                        return x;
                }
                else
                {
                        x=arr[front];
                        arr[front]=0;
                        front++;
                        return x;
                }
        }
        int count()
        {
                return (rear-front+1);
        }
        void display()
        {
                cout<<"The Queue consist of items are : "<<endl;
                for(int i=0;i<5;i++)
                {
                        cout<<arr[i]<<" ";
                }
        }
};

int main()
{
  queue q1;
  int option,value;
  do{


        cout<<"\n What operation you want do  in the Queue ? or press 0 to
Exit."<<endl;
        cout<<"1.Enqueue()"<<endl;
        cout<<"2.Dequeue()"<<endl;
        cout<<"3.isEmpty()"<<endl;
        cout<<"4.isFull()"<<endl;
        cout<<"5.Count()"<<endl;
        cout<<"6.Display()"<<endl;
        cout<<"7.Clear screen"<<endl;

        cout<<"Enter the option you want to do ? "<<endl;
        cin>>option;

        switch(option)
        {
                        case 0:
                                    break;
                        case 1:
                                    cout<<"Enqueue Function called: /n Enter the
value ? "<<endl;
                                    cin>>value;
                                    q1.enqueue(value);
                                    break;

                        case 2:
                                    cout<<"Dequeue Function called:"<<endl;
                                    q1.dequeue();
                                    break;
```

```
        case 3:
                if(q1.isEmpty())
                {
                        cout<<"Queue is Empty"<<endl;
                }
                else
                {
                        cout<<"Queue is not Empty"<<endl;
                }
                break;

        case 4:
                if(q1.isFull())
                {
                        cout<<"Queue is Full"<<endl;
                }
                else
                {
                        cout<<"Queue is not Full"<<endl;
                }
                break;

        case 5:
                cout<<"Count Function called /n The total
number of item is : "<<q1.count()<<endl;
                break;

        case 6:
                cout<<"Display is called :"<<endl;
```

```
                                q1.display();

                                break;


                    case 7:

                                system("cls");

                                break;


                    default:

                                cout<<"Enter correct option!!!"<<endl;

                                break;

            }



  }

  while(option!=0);

}
```

OUTPUT:

```
What operation you want do  in the Queue ? or press 0 to Exit.

1.Enqueue()

2.Dequeue()

3.isEmpty()

4.isFull()

5.Count()

6.Display()

7.Clear screen

Enter the option you want to do ?

6

Display is called :

The Queue consist of items are :
```

```
5   4 8 5 6
```

3. Write Code various operations on one way (singly) linked list.

INPUT:

```
#include<iostream>

using namespace std;

class Node{
  public:
            int key;
            int data;
            Node* next;

            Node()
            {
                    key=0;
                    data=0;
                    next=NULL;
            }

            Node(int k,int d)
            {
                    key = k;
                    data = d;
            }
};

class SinglyLinkedList{
```

```
public:
	Node* head;

	SinglyLinkedList()
	{
		head=NULL;
	}
	SinglyLinkedList(Node *n)
	{
			//point to the newly passed node
			head = n;
	}



	//1. Check if node exists using key value
	Node* nodeExists(int k)
	{
		Node* temp = NULL;

		Node* ptr = head;
		while(ptr!=NULL)
		{
			if(ptr->key==k)
			{
				temp=ptr;
			}
			ptr= ptr->next;
		}
		return temp;
	}
```

```
//2.Append a node to the list

void appendNode(Node *n)
{
        if(nodeExists(n->key)!=NULL)
        {
                cout<<"Node already exists with key value : "<<n->key<<".
Append another node with different key value"<<endl;
        }
        else
        {
                if(head==NULL)
                {
                        head = n;
                        cout<<"Node Append"<<endl;
                }
                else
                {
                        Node* ptr = head;
                        while(ptr->next!=NULL)
                        {
                                ptr = ptr->next;
                        }
                        ptr->next=n;
                        cout<<"Node Appended"<<endl;
                }
        }
}
```

```
//3. Prepend Node - Attach a node at the start
void prependNode(Node* n)
{
        if(nodeExists(n->key)!=NULL)
        {
                cout<<"Node already exists with key value : "<<n->key<<".
Append another node with different key value"<<endl;
        }
        else
        {
                n->next=head;
                head=n;
                cout<<"Node Prepended"<<endl;
        }
}


//4. insert a node after a particular node in the list
void insertNodeAfter(int k,Node *n)
{
        Node* ptr = nodeExists(k);
        if(ptr==NULL)
        {
                cout<<"No node exists with key value:" <<k<<endl;
        }
        else
        {
                if(nodeExists(n->key)!=NULL)
                {
                        cout<<"Node already exists with key value : "<<n->key<<".
Append another node with different key value"<<endl;
```

```
                }
                else
                {
                        n->next=ptr->next;
                        ptr->next=n;
                        cout<<"Node Inserted"<<endl;
                }
        }
}

//5. Delete node by unique key
void deleteNodeByKey(int k)
{
        if(head==NULL)
        {
                cout<<"singly linked list already Empty can't delete"<<endl;
        }
        else if(head!=NULL)
        {
                if(head->key==k)
                {
                        head = head->next;
                        cout<<"Node UNLINKED with keys value :"<<k<<endl;
                }
                else
                {
                        Node* temp=NULL;
                        Node* prevptr = head;
                        Node* currentptr = head->next;
                        while(currentptr!=NULL)
```

```
                    {
                        if(currentptr->key==k)
                        {
                            temp = currentptr;
                            currentptr=NULL;
                        }
                        else
                        {
                            prevptr = prevptr->next;
                            currentptr = currentptr->next;
                        }
                    }
                    if(temp!=NULL)
                    {
                        prevptr->next=temp->next;
                        cout<<"Node unlinked with key value : "<<k<<endl;
                    }
                    else
                    {
                        cout<<"Node doesn't exist with key value : 
"<<k<<endl;
                    }
                }
            }
        }

        //6. Update Node
        void updateNodeByKey(int k, int d)
        {
            Node* ptr = nodeExists(k);
```

```
		if(ptr!=NULL)
		{
			ptr->data=d;
			cout<<"Node data update successfully"<<endl;
		}
		else
		{
			cout<<"Node doesn't exist with key value :"<<k<<endl;
		}
	}

	//7. printing
	void printList()
	{
		if(head==NULL)
		{
			cout<<"No node in singly linked list";
		}
		else
		{
			cout<<endl<<"Singly linked list value :";
			Node* temp = head;

			while(temp!=NULL)
			{
				cout<<"("<<temp->key<<","<<temp->data<<") --> ";
				temp = temp->next;
			}
		}
	}
```

```
};



int main(){


  SinglyLinkedList s;
  int option;
  int key1,k1,data1;
  do
  {
        cout<<"\n What operation do you want to perform? Select option number.
Enter 0 to exit"<<endl;
        cout<<"1. appendNode()"<<endl;
        cout<<"2. prependNode()"<<endl;
        cout<<"3. insertNodeAfter()"<<endl;
        cout<<"4. deleteNodeByKey()"<<endl;
        cout<<"5. updateNodeByKey()"<<endl;
        cout<<"6. print()"<<endl;
        cout<<"7. Clear Screen"<<endl<<endl;

        cin>>option;
        Node* n1 = new Node();

        switch(option)
        {
              case 0:
                    break;
              case 1:
                    cout<<"Append node operation \n Enter key & data of the node
to be append"<<endl;
                    cin>>key1;
```

```
                        cin>>data1;

                        n1->key=key1;

                        n1->data=data1;

                        s.appendNode(n1);

                        break;


                case 2:

                        cout<<"Prepend node operation \n Enter key & data of the node
to be Prepend"<<endl;

                        cin>>key1;

                        cin>>data1;

                        n1->key=key1;

                        n1->data=data1;

                        s.prependNode(n1);

                        break;


                case 3:

                        cout<<"Insert node after operation \n Enter key of existing Node
after which you want to insert this new node"<<endl;

                        cin>>k1;

                        cout<<"Enter key & data of the new node first: "<<endl;

                        cin>>key1;

                        cin>>data1;

                        n1->key=key1;

                        n1->data=data1;

                        s.insertNodeAfter(k1,n1);

                        break;


        case 4:

                        cout<<"delete node by key operation \n Enter key of the node to
be deleted"<<endl;
```

```
                        cin>>k1;

                        s.deleteNodeByKey(k1);

                        break;


            case 5:

                        cout<<"update node by key operation \n Enter the existing key
and new data to be updated"<<endl;

                        cin>>key1;

                        cin>>data1;

                        s.updateNodeByKey(key1,data1);

                        break;


            case 6:

                        s.printList();

                        break;


            case 7:

                        system("cls");

                        break;

            default:

                  cout<<"Enter proper option number "<<endl;

            }

  }while(option!=0);


  return 0;

}
```

OUTPUT:

```
What operation do you want to perform? Select option number. Enter 0 to exit

1. appendNode()
```

```
2. prependNode()

3. insertNodeAfter()

4. deleteNodeByKey()

5. updateNodeByKey()

6. print()

7. Clear Screen


6


Singly linked list value :(1,45) --> (2,54) --> (3,77) -->
```

4.Write Code various operations on two way (doubly) linked list.

INPUT:

```
#include<iostream>
using namespace std;

class Node{
    public:
        int key;
        int data;
        Node* next;
        Node* previous;

        Node()
        {
            key=0;
            data=0;
            next=NULL;
```

```
                previous=NULL;
        }
        Node(int k,int d)
        {
                key=k;
                data=d;
                next=NULL;
                previous=NULL;
        }
};

class DoublyLinkedList{
    public:
        Node* head;
        DoublyLinkedList()
        {
                head=NULL;
        }
        DoublyLinkedList(Node *n)
        {
                head=n;
        }

        //check whether the node exist or not
        Node* CheckNodeExist(int k)
        {
                Node* temp=NULL;
                Node* ptr=head;
                while(ptr!=NULL)
                {
                        if(ptr->key==k)
                        {
                                temp=ptr;

                        }
                        ptr=ptr->next;
                }
                return temp;
        }

        void AppendNode(Node* n)
        {
                if(CheckNodeExist(n->key)!=NULL)
                {
                        cout<<"Node already exists with key "<<n->key<<" Append
another node with different key value"<<endl;
                }
                else{
                        if (head == NULL)
                        {
                head = n;
                cout << "Node Appended as Head node." << endl;
        }
        else{
                        Node* ptr=head;
                        while(ptr->next!=NULL)
                        {
                                ptr=ptr->next;
                        }
                        ptr->next=n;
                        n->previous=ptr;
                        cout<<"Node Appended successfully."<<endl;
                            }
                        }


                }

                // Prepend Node
                void PrependNode(Node* n)
                {
                    if(CheckNodeExist(n->key)!=NULL)
                    {
                        cout<<"Node already Exist "<<n->key<<" Append another node
with different key value"<<endl;
                    }
                    else
                    {
                        if(head==NULL)
                        {
                            head=n;
                            cout<<"Node Prepended as Head node"<<endl;
                        }
                        else
                        {
                            head->previous=n;
                            n->next=head;
                            head=n;
                            cout<<"Node Prepeneed"<<endl;
                        }
                    }
```

```
}


//Insert Node
void InsertNodeAfter(int k,Node* n)
{
        Node* ptr=CheckNodeExist(k);
        if(ptr==NULL)
        {
                cout<<"No Node Exist with key value"<<endl;
        }
        else
        {
                if(CheckNodeExist(n->key)!=NULL)
                {
                        cout<<"Node already Exist "<<n->key<<" Append another
node with different key value"<<endl;
                }
                else
                {
                        Node *nextNode=ptr->next;
                        //apending at the end
                        if(nextNode==NULL)
                        {
                                ptr->next=n;
                                n->previous=ptr;
                                cout<<"Node Inserted at the end"<<endl;
                        }
                        //appending in between
                        else
                        {
```

```
                                        n->next=nextNode;

                                        nextNode->previous=n;

                                        n->previous=ptr;

                                        ptr->next=n;

                                   }

                              }

                         }


                    }


                    //delete Node

                    void DeleteNodeByKey(int k)

                    {

                         Node* ptr=CheckNodeExist(k);

                         if(ptr==NULL)

                         {

                              cout<<"No Node Exist with key value"<<k<<endl;

                         }

                         else{

                              if(head==NULL)

                              {

                                   cout<<"DoublyLinkedList is already Empty ,can't
delete.'"<<endl;

                              }

                              else if(head!=NULL)

                              {

                                   if(head->key==k)

                                   {

                                        head=head->next;

                                        cout<<"Node Unlinked with key value "<<k<<endl;
```

```
                        }
                        else{
                                Node *nextNode=ptr->next;
                                Node *prevNode=ptr->previous;

                                //deleting at the end
                                if(nextNode==NULL)
                                {
                                        prevNode->next=NULL;
                                        cout<<"Node deleted at the end"<<endl;
                                }
                                else{
                                        prevNode->next=nextNode;
                                        nextNode->previous=prevNode;
                                        cout<<"Node Deletion in between"<<endl;
                                }

                        }
                }
        }
}

//Update node
void updateNode(int k,int d)
{
                Node* ptr=CheckNodeExist(k);
                if(ptr!=NULL)
                {
                        ptr->data=d;
                        cout<<"Node Updated Successfully"<<endl;
```

```
                    }
                    else
                    {
                        cout<<"Node doesnot Exist with key value "<<k<<endl;
                    }
            }

            //print
            void PrintList()
            {
                if(head==NULL)
                {
                    cout<<"No node in DoublyLinkedList"<<endl;
                }
                else
                {
                    cout<<endl<<"DoublyLinkedlist value : ";
                    Node* temp=head;
                    while(temp!=NULL)
                    {
                        cout<<"("<<temp->key<<","<<temp->data<<")<-->";
                        temp=temp->next;
                    }
                }
            }
};

int main()
{
    DoublyLinkedList obj;
```

```
	int option;
	int key1,data1,k1;

	do
	{
		cout<<"\n What operation do you want to perform? Select option number.
Enter 0 to exit"<<endl;
		cout<<"1. appendNode()"<<endl;
		cout<<"2. prependNode()"<<endl;
		cout<<"3. insertNodeAfter()"<<endl;
		cout<<"4. deleteNodeByKey()"<<endl;
		cout<<"5. updateNodeByKey()"<<endl;
		cout<<"6. print()"<<endl;
		cout<<"7. Clear Screen"<<endl<<endl;

		cin>>option;
		Node* n1 = new Node();

		switch(option)
		{
			case 0:
				break;

			case 1:
				cout<<"to append Node please provide key value and data of the
Node ."<<endl;
				cin>>key1;
				cin>>data1;
				n1->key=key1;
				n1->data=data1;
				obj.AppendNode(n1);
```

```
                        break;

                case 2:
                        cout<<"to prepend NOde please provide key value and data of
the Node ."<<endl;
                        cin>>key1;
                        cin>>data1;
                        n1->key=key1;
                        n1->data=data1;
                        obj.PrependNode(n1);
                        break;

                case 3:
                        cout<<"Enter the key value of node after which you want to
insert. ";
                        cin>>k1;
                        cout<<"to append Node please provide key value and data of the
Node ."<<endl;
                        cin>>key1;
                        cin>>data1;
                        n1->key=key1;
                        n1->data=data1;
                        obj.InsertNodeAfter(k1,n1);
                        break;

                case 4:
                            cout<<"Enter the key value of node which you want to
delete. ";
                            cin>>k1;
                            obj.DeleteNodeByKey(k1);
                            break;
                case 5:
                            cout<<"to Update Node please provide key value and data
of the Node ."<<endl;
                            cin>>key1;
                            cin>>data1;
                            obj.updateNode(key1,data1);
                            break;

                case 6:
                            obj.PrintList();
                            break;

                case 7:
                            system("cls");
                            break;

                default:
                        cout<<"Enter Correct option!!!"<<endl;
                        break;
                }
        }while(option!=0);
        return 0;
}
```

OUTPUT:

```
    What operation do you want to perform? Select option number. Enter 0 to exit
   1. appendNode()
   2. prependNode()
   3. insertNodeAfter()
```

```
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen

6

DoublyLinkedlist value : (3,55)<-->(1,54)<-->(85,4)<-->(2,85)<-->
```

5.Write Code various operations on circular linked list.

INPUT:

```
#include<iostream>

using namespace std;

class Node{
  public:
        int key;
        int data;
        Node* next;

        Node()
        {
            key=0;
            data=0;
            next=NULL;
        }
```

```
                Node(int k, int d)
                {
                        key=k;
                        data=d;
                }
};

class CircularLinkedList
{
  public:
                Node* head;

                CircularLinkedList()
                {
                        head = NULL;
                }

                //1. check if node exists using key value

                Node* nodeExists(int k)
                {
                        Node* temp = NULL;
                        Node* ptr = head;

                        if(ptr==NULL)
                        {
                                return temp;
                        }
                        else
                        {
```

```
                do
                {
                        if(ptr->key==k)
                        {
                                temp=ptr;
                        }
                        ptr = ptr->next;
                }
                while(ptr!=head);
                return temp;
        }
}



//2.Append a node to the list

void appendNode(Node *new_node)
{
        if(nodeExists(new_node->key)!=NULL)
        {
                cout<<"Node already exists with key value : "
                <<new_node->key
                <<". Appened another node with different key value"
                <<endl;
        }
        else
        {
                if(head==NULL)
                {
                        head = new_node;
                        new_node->next = head;
                        cout<<"Node Appened at first Head position"<<endl;
                }
                else
                {
                        Node* ptr = head;
                        while(ptr->next!=head)
                        {
                                ptr = ptr->next;
                        }
                        ptr->next=new_node;
                        new_node->next=head;
                        cout<<"Node Appended"<<endl;
                }
        }
}

//3. Prepend Node - Attach a node at the start

void prependNode(Node* new_node)
{
        if(nodeExists(new_node->key)!=NULL)
        {
                        cout<<"Node already exists with key value : "
                <<new_node->key
                <<". Append another node with different key value"
                <<endl;
        }
        else
        {
                if(head==NULL)
                {
                        head = new_node;
                        new_node->next=head;
                        cout<<"Node Prepend at first Head position"<<endl;
                }
                else
                {
                        Node* ptr = head;
                        while(ptr->next!=head)
                        {
                                ptr = ptr->next;
                        }
                        ptr->next=new_node;
                        new_node->next=head;
                        head=new_node;
                        cout<<"Node Prepend"<<endl;
                }
        }
}

//4. Insert a node after a particular node in the list

void insertNodeAfter(int k,Node *new_node)
{
        Node* ptr = nodeExists(k);
        if(ptr==NULL)
        {
                cout<<"No one Exists with key value of : "<<k<<endl;
        }
        else
        {
                if(nodeExists(new_node->key)!=NULL)
                {
                        cout<<"Node already exists with key value : "
                        <<new_node->key
                        <<". Appened another node with different key value"
                        <<endl;
                }
                else
                {
                        if(ptr->next==head)
                        {
                                new_node->next=head;
                                ptr->next=new_node;
                                cout<<"Node inserted at the end"<<endl;
                        }
                        else
                        {
                                new_node->next=ptr->next;
                                ptr->next=new_node;
                                cout<<"Node Inserted in between"<<endl;
                        }
                }
        }
}

// 5. Delete node by unique key

void deleteNodeByKey(int k)
```

```
        {
            Node* ptr = nodeExists(k);
            if(ptr==NULL)
            {
                cout<<"No node        exists with key value OF : "<<k
                <<endl;
            }
            else
            {
                if(ptr==head)
                {
                    if(head->next==NULL)
                    {
                        head=NULL;
                        cout<<"Head Node Unlinked.... List Empty";
                    }
                    else
                    {
                        Node* ptr1 = head;
                        while(ptr1->next!=head)
                        {
                            ptr1 = ptr1->next;
                        }
                        ptr1->next = head->next;
                        head = head->next;
                        cout<<"Node unlinked with key values :
"<<k<<endl;
                    }

                }
                else
                {
                        Node* temp=NULL;
                        Node* prevptr = head;
                        Node* currentptr = head->next;
                        while(currentptr!=NULL)
                        {
                                if(currentptr->key==k)
                                {
                                        temp =          currentptr;
                                        currentptr=NULL;
                                }
                                else
                                {
                                        prevptr = prevptr->next;
                                        currentptr = currentptr->next;
                                }
                        }
                        prevptr->next=temp->next;
                        cout<<"Node UNLINKED With Keys Values : "<<k<<endl;
                }
        }
}

//6. update node
void updateNodeByKey(int k, int d)
{
        Node* ptr = nodeExists(k);
        if(ptr!=NULL)
        {
```

```
                        ptr->data=d;

                        cout<<"Node data updated successfully"<<endl;

                }

                else

                {

                        cout<<"node doesn't exists with key value : "<<k<<endl;

                }

        }



        //7. printing


        void printList()

        {

                if(head==NULL)

                {

                        cout<<"No Node In CircularLinkedList LinkedList";

                }

                else

                {

                        cout<<endl<<"head address : "<<head<<endl;

                        cout<<"CircularLinkedList Linkedlist Values : "<<endl;


                        Node* temp = head;


                        do

                        {

                                cout<<"("<<temp->key<<","<<temp->data<<","<<temp-
>next<<") --> ";

                                temp = temp->next;
                        }
                        while(temp!=head);
                }
        }
};

int main(){

  CircularLinkedList obj;
  int option;
  int key1,k1,data1;
  do
  {
        cout<<"\n What operation do you want to perform? Select option number.
Enter 0 to exit"<<endl;
        cout<<"1. appendNode()"<<endl;
        cout<<"2. prependNode()"<<endl;
        cout<<"3. insertNodeAfter()"<<endl;
        cout<<"4. deleteNodeByKey()"<<endl;
        cout<<"5. updateNodeByKey()"<<endl;
        cout<<"6. print()"<<endl;
        cout<<"7. Clear Screen"<<endl<<endl;

        cin>>option;
        Node* n1 = new Node();

        switch(option)
        {
                case 0:
                        break;
                case 1:
                        cout<<"Append node operation \n Enter key & data of the node
to be append"<<endl;
                        cin>>key1;
                        cin>>data1;
                        n1->key=key1;
                        n1->data=data1;
                        obj.appendNode(n1);
                        break;

                case 2:
                        cout<<"Prepend node operation \n Enter key & data of the node
to be Prepend"<<endl;
                        cin>>key1;
                        cin>>data1;
                        n1->key=key1;
                        n1->data=data1;
                        obj.prependNode(n1);
                        break;

                case 3:
                        cout<<"Insert node after operation \n Enter key of existing Node
after which you want to insert this new node"<<endl;
                        cin>>k1;
                        cout<<"Enter key & data of the new node first: "<<endl;
                        cin>>key1;
                        cin>>data1;
                        n1->key=key1;
                        n1->data=data1;
                        obj.insertNodeAfter(k1,n1);
                        break;
```

```
            case 4:
                                cout<<"delete node by key operation \n Enter key of the node to
be deleted"<<endl;
                                cin>>k1;
                                obj.deleteNodeByKey(k1);
                                break;

            case 5:
                                cout<<"update node by key operation \n Enter key value and
new data to be updated"<<endl;
                                cin>>key1;
                                cin>>data1;
                                obj.updateNodeByKey(key1,data1);
                                break;

            case 6:
                                obj.printList();
                                break;

            case 7:
                                system("cls");
                                break;
            default:
                      cout<<"Enter proper option number "<<endl;
            }
  }while(option!=0);

  return 0;
}
```

OUTPUT:

```
What operation do you want to perform? Select option number. Enter 0 to exit
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen

6

head address : 0x8815a0
CircularLinkedList Linkedlist Values :
(3,88,0x881560) --> (1,54,0x881580) --> (2,55,0x8815a0) -->
```