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

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

            Node(int v)
            {
                  value=v;
                  prev=next=nullptr;
            };

      };

      Node *head;
      Node *tail;
public:

      SentinelCircularList()
      {
            Node  *h=new Node(0);
            head=h;
            tail=h;

      }

      void addToBack(int value)
      {
            Node *newNode=new Node(value);

            newNode->next=head;
            newNode->prev=tail;
            tail->next=newNode;
            head->prev=newNode;
            tail=newNode;
      }

      void printforward()
      {
            if (nullptr==head->next)
                  cout << "list is empty" << endl;
            else
                  for(Node *current=head->next;current!
=head;current=current->next)
                        cout << current->value << endl;
      }

      void printbackward()
      {

            for(Node *current=head->prev;current!=head;current=current-
>prev)
                  cout << current->value << endl;
      }

      void addtofront(int value)
      {
            Node *newNode=new Node(value);

            newNode->next=head->next;
            newNode->prev=head;
            head->next->prev=newNode;
            head->next=newNode;

      }

      bool removeall()
      {
            Node *temp , *current = nullptr;
            for(temp=head->next;temp!=head->prev;temp=current)
            {
                  current=temp->next;
                  delete temp;
            }
            head->next=head->prev=nullptr;


            return false;
      }

      bool InsertAfter(int search,int value)
      {
            for(Node *current=head->next;current!=head;current=current-
>next)
            {
                  if(search==current->value)
                  {
                        Node *newNode=new Node(value);

                        newNode->prev=current;
                        newNode->next=current->next;
                        current->next->prev=newNode;
                        current->next=newNode;
                        return true;
                  }
            }
            return false;
      }

      bool removeNode(int value)
      {
            Node *temp;
            for(Node *current=head->next;current!=head;current=temp)
            {
                  if(value==current->value)
                  {
                        temp=current->next;
                        current->prev->next=current->next;
                        current->next->prev=current->prev;
                        delete current;
                        return true;
                  }
                  temp=current->next;
            }
            return false;
      }

      bool insertBefore(int search,int value)
      {
            for(Node *current=head->next;current!=head;current=current-
>next)
            {
                  if (search == current->value)
                  {
                        Node *newNode = new Node(value);

                        newNode->next = current;
                        newNode->prev = current->prev;
                        current->prev->next = newNode;
                        current->prev=newNode;
                        return true;
                  }
            }
            return false;
      }
};


int  main()
{
      int num,value;
      SentinelCircularList s1;

      while (cout << "enter value (enter 0 to stop)" << endl, cin >> num,
num)
      {
            s1.addToBack(num);
      }
      cout << "Printing forwrd" << endl;
      s1.printforward();
      cout << "Print Backward" << endl;
      s1.printbackward();

      while (cout << "enter the value to add in front (enter 0 to stop)"
<< endl, cin >> num, num)
      {
            s1.addtofront(num);
      }
      s1.printforward();

      while (cout << "enter the value before to add new value (enter 0 to
stop)" << endl, cin >> num, num)
      {
            cout << "enter value you want to add"<<endl;
            cin >> value;
            s1.insertBefore(num,value);
      }
      s1.printforward();

      while (cout << "enter the value after  to add new value (enter 0 to
stop)" << endl, cin >> num, num)
      {
            cout << "enter value you want to add"<<endl;
            cin >> value;
            s1.InsertAfter(num,value);
      }
      s1.printforward();

      while (cout << "enter the value to remove (enter 0 to stop)" <<
endl, cin >> num, num)
      {
            s1.removeNode(num);
      }
      s1.printforward();

      cout << "remove all node" << endl;
      s1.removeall();
      s1.printforward();
}
```