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

class CircularList
{
	class Node
	{
	public:
		Node *prev,*next;
		int value;
		Node (int value)
		{
			this->value = value;
		}
	};

	Node *start;
public:
	int cnt;
public:
	CircularList()
	{
		start = nullptr;
		cnt = 0;
	}


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

	if (nullptr == start)
	{
		start = newNode;
		newNode->next = newNode->prev = newNode;
	}
	else
	{
		newNode->next = start;
		newNode->prev = start->prev;
		start->prev->next = newNode;
		start->prev = newNode;
	}
	cnt++;

}

void  printForward()
{
	int i = 0;

	if(start == nullptr)
	{
		cout<<"list is empty"<<endl;
```

```
	}
	else
	{
		for (Node *cur=start; i<cnt; cur = cur->next)
		{
			i++;
			cout << cur->value << endl;
		}
	}
}


void printBackward()
{
	int i = 0;
	for (Node *cur = start->prev; i<cnt; cur = cur->prev)
	{
		i++;
		cout << cur->value << endl;
	}
}

void addToFront(int value)
{
	Node *newNode = new Node(value);
	cnt++;
	if (nullptr == start)
	{
		start = newNode;
		newNode->next = newNode->prev = newNode;
	}
	else
	{
		newNode->next = start;
		newNode->prev = start->prev;
		start->prev->next = newNode;
		start->prev = newNode;
		start = newNode;
	}
}


bool insertBefore(int search,int value)
{
	int i = 0;
	for (Node *cur = start; i<cnt; cur = cur->next)
	{
		i++;
		if(search == cur->value)
		{
			Node *newNode = new Node(value);
			cnt++;
			newNode->next = cur;
			newNode->prev = cur->prev;
```

```
                cur->prev->next = newNode;
                cur->prev = newNode;
                return true;
            }
        }

        return false;
}

bool insertAfter(int search,int value)
{
        int i = 0;
        for(Node *cur = start; i<cnt; cur = cur->next)
        {
            if(search == cur->value)
            {
                Node *newNode = new Node(value);
                cnt++;
                newNode->prev=cur;
                newNode->next = cur->next;
                cur->next->prev = newNode;
                cur->next = newNode;
                return true;
            }
        }
        return false;
}

void removeAll()
{
        Node *temp = nullptr;
        int i = 0;
        while (i < cnt-2)
        {
            i++;
            temp = start;
            start = start->next;
            delete temp;
        }
        start = nullptr;
        cnt = 0;
}




bool removeNode(int value)
{
        int i = 0;
        for (Node *cur = start; i<cnt; cur = cur->next)
        {
            if(value == cur->value)
            {
                if(cur == start)
```

```
                {
                      cur->next->prev = cur->prev;
                      cur->prev->next = cur->next;
                      start = cur->next;
                }
                else
                {
                      cur->next->prev = cur->prev;
                      cur->prev->next = cur->next;
                }
                delete cur;
                cnt--;
                return true;
          }
      }
      return false;
 }

};

int main()
{

      CircularList c1;
      int num,value;

      while (cout << "enter value (enter 0 to stop)" << endl, cin >> num,
num)
      {
            c1.addToBack(num);
      }

      cout << "Printing forwrd" << endl;
      c1.printForward();

      cout << "Print Backward" << endl;
      c1.printBackward();

      while (cout << "enter the value add in front (enter 0 to stop)" <<
endl, cin >> num, num)
      {
            c1.addToFront(num);
      }
      c1.printForward();

      while (cout << "enter the value to add new value (enter 0 to stop)"
<< endl, cin >> num, num)
      {
            cout << "enter value to add"<<endl;
            cin >> value;
            c1.insertBefore(num,value);
      }
      c1.printForward();
```

```
      while (cout << "enter the value (enter 0 to stop)" << endl, cin >>
num, num)
      {
            cout << "enter value  to add"<<endl;
            cin >> value;
            c1.insertAfter(num,value);
      }
      c1.printForward();

      while (cout << "enter the value   (enter 0 to stop)" << endl, cin
>> num, num)
      {
            c1.removeNode(num);
      }
      c1.printForward();

      cout << "remove all node" << endl;
      c1.removeAll();
      c1.printForward();

}
```