#include <iostream>

using namespace std;

struct Node

{

string data;

Node\* previous;

Node\* next;

};

class DoubleLinkedList

{

private:

Node\* front;

Node\* end;

public:

DoubleLinkedList()

{

front = NULL;

end = NULL;

}

DoubleLinkedList(Node\* front, Node\* end)

{

this->front = front;

this->end = end;

}

void forward\_traverse();

void backward\_traverse();

void add\_front(string);

void add\_after(Node\*, string);

void delete\_node(Node\*);

// Task

void add\_end(string);

void delete\_first();

void delete\_last();

void delete\_by\_data(string);

void add\_block(DoubleLinkedList);

};

void DoubleLinkedList::forward\_traverse()

{

Node\* temp = front;

while (temp != NULL)

{

cout << temp->data << endl;

temp = temp->next;

}

cout << endl;

}

void DoubleLinkedList::backward\_traverse()

{

Node\* temp = end;

while (temp != NULL)

{

cout << temp->data << endl;

temp = temp->previous;

}

cout << endl;

}

void DoubleLinkedList::add\_front(string data)

{

Node\* newnode = new Node;

newnode->data = data;

newnode->previous = NULL;

newnode->next = front;

// If list is empty

if (front == NULL)

{

end = newnode;

}

else

{

front->previous = newnode;

}

front = newnode;

}

void DoubleLinkedList::add\_after(Node\* n, string data)

{

Node\* newnode = new Node;

newnode->data = data;

newnode->previous = n;

newnode->next = n->next;

n->next->previous = newnode;

n->next = newnode;

if (n->next == NULL)

{

end = newnode;

}

}

void DoubleLinkedList::add\_end(string data)

{

Node\* newnode = new Node;

newnode->data = data;

newnode->previous = end;

newnode->next = NULL;

if (end != NULL)

{

end->next = newnode;

}

end = newnode;

if (front == NULL)

{

front = newnode;

}

}

void DoubleLinkedList::delete\_node(Node\* n)

{

if (n->previous == NULL)

{

front = n->next;

front->previous = NULL;

}

else if (n->next == NULL)

{

end = n->previous;

end->next = NULL;

}

else

{

n->previous->next = n->next;

n->next->previous = n->previous;

}

delete n;

n = nullptr;

}

void DoubleLinkedList::delete\_first()

{

if (front->next != NULL)

{

front = front->next;

}

delete front->previous;

front->previous = nullptr;

}

void DoubleLinkedList::delete\_last()

{

if (end->previous != NULL)

{

end = end->previous;

}

delete end->next;

end->next = nullptr;

}

void DoubleLinkedList::delete\_by\_data(string data)

{

Node\* temp = front;

while (temp->data != data)

{

temp = temp->next;

}

delete\_node(temp);

}

void DoubleLinkedList::add\_block(DoubleLinkedList other\_list)

{

this->end->next = other\_list.front;

other\_list.front->previous = this->end;

this->end = other\_list.end;

}

int main()

{

Node\* first = new Node;

Node\* second = new Node;

Node\* third = new Node;

first->data = "Apple";

first->previous = NULL;

first->next = second;

second->data = "Tissot";

second->previous = first;

second->next = third;

third->data = "Rolex";

third->previous = second;

third->next = NULL;

DoubleLinkedList mylist(first, third);

//mylist.backward\_traverse();

//mylist.forward\_traverse();

//mylist.add\_front("Porshce");

//mylist.backward\_traverse();

//mylist.forward\_traverse();

//mylist.add\_after(second,"SALAM");

//mylist.backward\_traverse();

//mylist.forward\_traverse();

//mylist.delete\_node(second);

//mylist.forward\_traverse();

//mylist.backward\_traverse();

//mylist.add\_end("HELLO");

//mylist.forward\_traverse();

//mylist.backward\_traverse();

//mylist.delete\_first();

//mylist.forward\_traverse();

//mylist.backward\_traverse();

//mylist.delete\_last();

//mylist.forward\_traverse();

//mylist.backward\_traverse();

//mylist.delete\_by\_data("Tissot");

//mylist.forward\_traverse();

//mylist.backward\_traverse();

Node\* otherfirst = new Node;

Node\* othersecond = new Node;

Node\* otherthird = new Node;

DoubleLinkedList othermylist(otherfirst, otherthird);

otherfirst->data = "BMW";

otherfirst->previous = NULL;

otherfirst->next = othersecond;

othersecond->data = "MERCEDES";

othersecond->previous = otherfirst;

othersecond->next = otherthird;

otherthird->data = "C++";

otherthird->previous = othersecond;

otherthird->next = NULL;

mylist.add\_block(othermylist);

mylist.forward\_traverse();

mylist.backward\_traverse();

return 0;

}