#include <iostream>

using namespace std;

struct node {

string data;

node\* prev;

node\* next;

};

class DDL {

node\* front;

node\* end;

public:

DDL()

{

front = NULL;

end = NULL;

}

DDL(node\* front, node\* end)

{

this->front = front;

this->end = end;

}

void forward\_traverse();

void backward\_traverse();

void add\_front(string);

void add\_end(string);

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

void delete\_node(node\*);

};

void DDL::forward\_traverse() {

node\* temp = front;

while (temp != NULL)

{

cout << temp->data << endl;

temp = temp->next;

}cout << endl;

}

void DDL::backward\_traverse() {

node\* temp = end;

while (temp != NULL)

{

cout << temp->data << endl;

temp = temp->prev;

}

}

void DDL::add\_front(string data) {

node\* new\_node = new node;

new\_node->data = data;

new\_node->prev = NULL;

new\_node->next = front;

//if list is empty

if (front == NULL) {

end = new\_node;

}//if list is not empty

else {

front->prev = new\_node;

}

front = new\_node;

}

void DDL::add\_after(node\* n, string data) {

node\* new\_node = new node;

new\_node->data = data;

new\_node->prev = n;

new\_node->next = n->next;

n->next->prev = new\_node;

n->next = new\_node;

if (n->next == NULL) {

end = new\_node;

}

}

void DDL::delete\_node(node\* n) {

//it means n is first

if (n->prev == NULL) {

front = n->next;

front->prev = NULL;

}

//it means n is end

else if (n->next == NULL) {

end = n->prev;

end->next = NULL;

}

else {

n->prev->next = n->next;

n->next->prev = n->prev;

}

delete n;

n = nullptr;

}

void main() {

node\* first = new node;

node\* second = new node;

node\* third = new node;

first->data = "Apple";

first->prev = NULL;

first->next = second;

second->data = "Tissot";

second->prev = first;

second->next = third;

third->data = "Rolex";

third->prev = second;

third->next = NULL;

DDL mylist(first, third);

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

/\*mylist.forward\_traverse();

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

mylist.forward\_traverse();\*/

//mylist.backward\_traverse();

/\*mylist.backward\_traverse();

cout << endl;

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

mylist.backward\_traverse();\*/

//mylist.delete\_node(second);

//mylist.forward\_traverse();

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

//delete\_first

//delete\_last

// deleteByData("Rolex");

/\*addBlock(first, last);

1 2 3=>mylist

mylist2=>55 66 77

mylist.AddBlock(mylist2.front,mylist2.end)\*/

}