#include <iostream>

using namespace std;

template<class T>

struct DNode

{

public:

T value;

DNode \*prev;

DNode \*next;

public:

DNode() { }

DNode(T t, DNode \*prev, DNode \*next) {

this->value = t;

this->prev = prev;

this->next = next;

}

};

template<class T>

class DoubleLink

{

public:

DoubleLink();

~DoubleLink();

int size();

int is\_empty();

T get(int index);

T get\_first();

T get\_last();

int insert(int index, T t);

int insert\_first(T t);

int append\_last(T t);

int del(int index);

int delete\_first();

int delete\_last();

private:

int count;

DNode<T> \*phead;

private:

DNode<T> \*get\_node(int index);

};

template<class T>

DoubleLink<T>::DoubleLink()

{

// 创建“表头”。注意：表头没有存储数据！

phead = new DNode<T>();

phead->prev = phead->next = phead;

// 设置链表计数为0

count = 0;

}

// 析构函数

template<class T>

DoubleLink<T>::~DoubleLink()

{

// 删除所有的节点

DNode<T>\* ptmp;

DNode<T>\* pnode = phead->next;

while (pnode != phead)

{

ptmp = pnode;

pnode = pnode->next;

delete ptmp;

}

// 删除"表头"

delete phead;

phead = NULL;

}

// 返回节点数目

template<class T>

int DoubleLink<T>::size()

{

return count;

}

// 返回链表是否为空

template<class T>

int DoubleLink<T>::is\_empty()

{

return count == 0;

}

// 获取第index位置的节点

template<class T>

DNode<T>\* DoubleLink<T>::get\_node(int index)

{

// 判断参数有效性

if (index<0 || index >= count)

{

cout << "get node failed! the index in out of bound!" << endl;

return NULL;

}

// 正向查找

if (index <= count / 2)

{

int i = 0;

DNode<T>\* pindex = phead->next;

while (i++ < index) {

pindex = pindex->next;

}

return pindex;

}

// 反向查找

int j = 0;

int rindex = count - index - 1;

DNode<T>\* prindex = phead->prev;

while (j++ < rindex) {

prindex = prindex->prev;

}

return prindex;

}

// 获取第index位置的节点的值

template<class T>

T DoubleLink<T>::get(int index)

{

return get\_node(index)->value;

}

// 获取第1个节点的值

template<class T>

T DoubleLink<T>::get\_first()

{

return get\_node(0)->value;

}

// 获取最后一个节点的值

template<class T>

T DoubleLink<T>::get\_last()

{

return get\_node(count - 1)->value;

}

// 将节点插入到第index位置之前

template<class T>

int DoubleLink<T>::insert(int index, T t)

{

if (index == 0)

return insert\_first(t);

DNode<T>\* pindex = get\_node(index);

DNode<T>\* pnode = new DNode<T>(t, pindex->prev, pindex);

pindex->prev->next = pnode;

pindex->prev = pnode;

count++;

return 0;

}

// 将节点插入第一个节点处。

template<class T>

int DoubleLink<T>::insert\_first(T t)

{

DNode<T>\* pnode = new DNode<T>(t, phead, phead->next);

phead->next->prev = pnode;

phead->next = pnode;

count++;

return 0;

}

// 将节点追加到链表的末尾

template<class T>

int DoubleLink<T>::append\_last(T t)

{

DNode<T>\* pnode = new DNode<T>(t, phead->prev, phead);

phead->prev->next = pnode;

phead->prev = pnode;

count++;

return 0;

}

// 删除index位置的节点

template<class T>

int DoubleLink<T>::del(int index)

{

DNode<T>\* pindex = get\_node(index);

pindex->next->prev = pindex->prev;

pindex->prev->next = pindex->next;

delete pindex;

count--;

return 0;

}

// 删除第一个节点

template<class T>

int DoubleLink<T>::delete\_first()

{

return del(0);

}

// 删除最后一个节点

template<class T>

int DoubleLink<T>::delete\_last()

{

return del(count - 1);

}

// 双向链表操作int数据

void int\_test()

{

int iarr[4] = { 10, 20, 30, 40 };

cout << "----int\_test----" << endl;

// 创建双向链表

DoubleLink<int>\* pdlink = new DoubleLink<int>();

pdlink->insert(0, 20); // 将 20 插入到第一个位置

pdlink->append\_last(10); // 将 10 追加到链表末尾

pdlink->insert\_first(30); // 将 30 插入到第一个位置

// 双向链表是否为空

cout << "is\_empty()=" << pdlink->is\_empty() << endl;

// 双向链表的大小

cout << "size()=" << pdlink->size() << endl;

// 打印双向链表中的全部数据

int sz = pdlink->size();

for (int i = 0; i<sz; i++)

cout << "pdlink(" << i << ")=" << pdlink->get(i) << endl;

}

int main()

{

int\_test(); // 演示向双向链表操作“int数据”。

return 0;

}