**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <stdbool.h>

**typedef** **int** Info;

**typedef** **struct** {

Info info;

} Item;

**typedef** Item\* ItemRef;

**typedef** **struct** NodeStruct\* NodeRef;

**typedef** **struct** NodeStruct {

ItemRef item;

NodeRef next;

NodeRef prev;

} Node;

**typedef** **struct** {

NodeRef head;

NodeRef tail;

**int** size;

} Dlist;

**typedef** Dlist\* DlistRef;

ItemRef **initializeItem**(Info g);

/\* allocates storage to store one item and sets it to value g \*/

NodeRef **initializeNode**(Info g);

/\* allocates storage to store one node with item value g and NULL links pointers \*/

DlistRef **initializeDlist**(**void**);

/\* allocates storage to for a list header with NULL head and tail and item count 0 \*/

**void** **printListForward**(DlistRef dl);

/\* print list items in forward order starting at head \*/

/\* title can be used to add a descriptive title for the list output \*/

**void** **printListBackward**(DlistRef dl);

/\* print list items in backward order starting at tail \*/

/\* title can be used to add a descriptive title for the list output \*/

**void** **insertNodeAtHead**(DlistRef dl, Info g);

/\* insert a new node with value g at the beginning of the list \*/

**void** **insertNodeAtTail**(DlistRef dl, Info g);

/\* insert a new node at the end of the list with value g \*/

NodeRef **findNode**(DlistRef dl, Info g);

/\* find the node with value g in the list and return its reference \*/

/\* returns NULL if not found \*/

**void** **insertNodeAt**(DlistRef dl, Info oldval, Info newval);

/\* insert a new node with value newval after the node with value oldval \*/

/\* if oldval is not found in the list, insert newval at the end of the list \*/

/\* use findNode() to implement this function \*/

**void** **updateNode**(DlistRef dl, Info oldval, Info newval);

/\* find the first node with value oldval and update the value to newval \*/

/\* use findNode() to implement this function \*/

Info **sumAllNodes**(DlistRef dl);

/\* computes the sum of the Info values of all the nodes in the list \*/

**void** **deallocateNode**(NodeRef x);

/\* deallocate storage for node x and the item it points to \*/

bool **deleteNodeAtHead**(DlistRef dl, Info\* g);

/\* delete node at the head of the list including deallocating the space \*/

/\* for the node and the item it points to by calling deallocateNode \*/

/\* return false if list is empty and true otherwise \*/

/\* return value of deleted item in g \*/

bool **deleteNodeAtTail**(DlistRef dl, Info\* g);

/\* delete node at the tail of the list including deallocating the space \*/

/\* for the node and the item it points to by calling deallocateNode \*/

/\* return false if list is empty and true otherwise \*/

/\* return value of deleted item in g \*/

bool **deleteNodeAt**(DlistRef dl, Info g);

/\* delete the first node with value g in the list \*/

/\* return false if list is empty or item is not found; true otherwise \*/

ItemRef **initializeItem**(Info g){

ItemRef item = **malloc**(**sizeof**(Item));

item->info = g;

**return** item;

}

NodeRef **initializeNode**(Info g){

NodeRef node = **malloc**(**sizeof**(Node));

node->item = initializeItem(g);

node->prev = NULL;

node->next = NULL;

**return** node;

}

DlistRef **initializeDlist**(**void**){

DlistRef list = **malloc**(**sizeof**(Dlist));

list->head = NULL;

list->size = 0;

list->tail = NULL;

**return** list;

}

**void** **insertNodeAtHead**(DlistRef dl, Info g)

{

NodeRef NewNode = initializeNode (g);

NewNode->next = dl->head;

**if**(NewNode->next == NULL)

{

NewNode->next = NULL;

NewNode->prev = NULL;

dl->head = NewNode;

dl->tail = NewNode;

}

**else**

{

dl->head = NewNode;

NewNode->prev = NULL;

NewNode->next->prev = NewNode;

}

}

**void** **printListForward**(DlistRef dl){

NodeRef tempNode = dl->head;

**while**(tempNode != NULL){

**printf**(" %d",tempNode->item->info);

tempNode = tempNode->next;

}

**printf**("\n");

}

**void** **printListBackward**(DlistRef dl){

NodeRef tempNode = dl->tail;

**while**(tempNode != NULL){

**printf**( " %d", tempNode->item->info);

tempNode = tempNode->prev;

}

**printf**("\n");

}

**void** **insertNodeAtTail**(DlistRef dl, Info g){

NodeRef NewNode = initializeNode (g);

NewNode->prev = dl->tail;

**if**(NewNode->prev == NULL)

{

NewNode->prev = NULL;

NewNode->next = NULL;

dl->tail = NewNode;

dl->head = NewNode;

}

**else**

{

dl->tail = NewNode;

NewNode->next = NULL;

NewNode->prev->next = NewNode;

}

}

NodeRef **findNode**(DlistRef dl, Info g){

NodeRef point = dl->head;

**while**(point->item->info != g && point->next != NULL){

point = point->next;

}

**if**( point->next == NULL){

point = NULL;

}

**return** point;

}

**void** **insertNodeAt**(DlistRef dl, Info oldval, Info newval){

NodeRef newNode = initializeNode(newval);

NodeRef find = findNode(dl, oldval);

newNode->next = find->next;

newNode->prev = find;

newNode->next->prev = newNode;

newNode->prev->next = newNode;

}

**void** **updateNode**(DlistRef dl, Info oldval, Info newval){

NodeRef point = findNode (dl,oldval);

**if** (point == NULL){

point->item->info = newval;

}

}

Info **sumAllNodes**(DlistRef dl){

Info sum = 0;

NodeRef point = dl->head;

**while**(point->next != NULL)

{

sum += point->item->info;

point = point->next;

}

sum += point->item->info;

**printf**(" Sum = %d\n", sum);

**return** sum;

}

**void** **deallocateNode**(NodeRef x){

**free** (x->item);

**free** (x);

}

bool **deleteNodeAtHead**(DlistRef dl, Info\* g){

NodeRef deleteNode = dl->head;

**if**(deleteNode == NULL){

**return** false;

}

deleteNode->next->prev = NULL;

dl->head = deleteNode->next;

g = &deleteNode->item->info;

deallocateNode(deleteNode);

**return** g;

}

bool **deleteNodeAtTail**(DlistRef dl, Info\* g){

NodeRef deleteEndNode = dl->tail;

**if**(deleteEndNode == NULL){

**return** false;

}

deleteEndNode->prev->next = NULL;

dl->tail = deleteEndNode->prev;

g = &deleteEndNode->item->info;

deallocateNode(deleteEndNode);

**return** g;

}

bool **deleteNodeAt**(DlistRef dl, Info g){

NodeRef find = findNode(dl, g);

**if**(find == NULL){

**return** false;

}

find->prev->next = find->next;

find->next->prev = find->prev;

find->prev = NULL;

find->prev = NULL;

deallocateNode(find);

**return** true;

}

**int** **main**(**void**)

{

DlistRef myList = initializeDlist();

Info n;

**for**(n = 1; n<=15; n++)

{

**if**(n%2 != 0)

{

insertNodeAtTail(myList, n);

}

}

printListForward(myList);

printListBackward(myList);

insertNodeAtTail( myList, 17);

insertNodeAtHead( myList, 0);

printListForward( myList);

insertNodeAt(myList,5,6);

insertNodeAt(myList, 7,8);

insertNodeAt(myList,9,10);

printListForward(myList);

sumAllNodes(myList);

Info g = NULL;

deleteNodeAtHead(myList,g);

deleteNodeAtHead(myList,g);

deleteNodeAtHead(myList,g);

deleteNodeAtTail(myList,g);

deleteNodeAtTail(myList,g);

deleteNodeAt(myList, 7);

printListForward(myList);

**return** EXIT\_SUCCESS;

}