**LINKEDLIST IMPLEMENTATION (Doubly linked using Structure)**

**/\*** linkedList**.**c**: LinkedList implementation in C. \*/**

#include "linkedList.h"

#include <assert.h>

#include <stdlib.h>

#include <stdio.h>

/\* Double Link\*/

struct DLink {

TYPE value;

struct DLink \* next;

struct DLink \* prev;

};

/\* Double Linked List with Head and Tail Sentinels \*/

struct linkedList{

int size;

struct DLink \*firstLink;

struct DLink \*lastLink;

};

/\*

initList

param lst the linkedList

pre: lst is not null

post: lst size is 0

\*/

void \_initList (struct linkedList \*lst) {

assert (!EQ(lst, 0));

lst->firstLink = malloc(sizeof(struct DLink));

assert(lst->firstLink != 0);

lst->lastLink = malloc(sizeof(struct DLink));

assert(lst->lastLink);

lst->firstLink->next = lst->lastLink;

lst->lastLink->prev = lst->firstLink;

lst->size = 0;

}

/\*

createList

param: none

pre: none

post: firstLink and lastLink reference sentinels

\*/

struct linkedList \*createLinkedList()

{

struct linkedList \*newList = malloc(sizeof(struct linkedList));

\_initList(newList);

return(newList);

}

/\*

\_addLinkBeforeBefore

param: lst the linkedList

param: l the link to add before

param: v the value to add

pre: lst is not null

pre: l is not null

post: lst is not empty

\*/

/\*

\_addLink

Funtion to add a value v to the list before the link l

param: lst the list

param: lnk the link to add before

param: v the value to add

pre: lst is not null

pre: lnk is not null

post: lst is not empty

\*/

void \_addLink(struct linkedList \*lst, struct DLink \*lnk, TYPE v)

{

assert (!EQ(lst, 0));

assert (!EQ(lnk, 0));

struct DLink \*newLink = (struct DLink \*)malloc(sizeof(struct DLink));

assert (!EQ(newLink, 0));

newLink->value = v;

newLink->next = lnk;

newLink->prev = lnk->prev;

lnk->prev->next = newLink;

lnk->prev = newLink;

lst->size++;

}

/\* Adds Before the provided link, l \*/

void \_addLinkBefore(struct linkedList \*lst, struct DLink \*l, TYPE v)

{

/\* Make sure the list is not empty \*/

assert (!EQ(l, 0));

/\* Construct a new dlink to be added \*/

struct DLink \*newLink = malloc(sizeof(struct DLink));

assert(newLink != 0);

assert (!EQ(lst, 0));

newLink->value = v;

/\* Change pointer on the argument dlink prev dlink \*/

l->prev->next = newLink;

/\* Change the prev of the newLink to point at former previous of the argument dlink \*/

newLink->prev = l->prev;

/\* Point the newLink at the argument dlink and vice-versa \*/

newLink->next = l;

l->prev = newLink;

lst->size++;

}

/\*

\_removeLink

param: lst the linkedList

param: l the linke to be removed

pre: lst is not null

pre: l is not null

post: lst size is reduced by 1

\*/

void \_removeLink(struct linkedList \*lst, struct DLink \*l)

{

/\* Make sure the list is not empty \*/

assert (!EQ(lst, 0));

assert(!isEmptyList(lst));

/\* Adjust two pointers to go around the outgoing dlink \*/

l->prev->next = l->next;

l->next->prev = l->prev;

/\* Free memory from link \*/

free(l);

lst->size--;

}

/\*

isEmptyList

param: lst the linkedList

pre: lst is not null

post: none

\*/

int isEmptyList(struct linkedList \*lst) {

return lst->size == 0;

}

/\* De-allocate all links of the list

param: lst pointer to the linked list

pre: none

post: All links (including the two sentinels) are de-allocated

\*/

void freeLinkedList(struct linkedList \*lst)

{

while(!isEmptyList(lst)) {

/\* remove the link right after the first sentinel \*/

\_removeLink(lst, lst->firstLink->next);

}

/\* remove the first and last sentinels \*/

free(lst->firstLink);

free(lst->lastLink);

}

/\* Deallocate all the links and the linked list itself.

param: v pointer to the dynamic array

pre: v is not null

post: the memory used by v->data is freed

\*/

void deleteLinkedList(struct linkedList \*lst)

{

assert (lst != 0);

freeLinkedList(lst);

free(lst);

}

/\* Function to print list

Pre: lst is not null

\*/

void \_printList(struct linkedList\* lst)

{

assert (!EQ(lst, 0));

struct DLink \*temp = lst->firstLink->next;

while (!EQ(lst->lastLink, temp)){

printf("%d\n", temp->value);

temp = temp->next;

}

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Deque Interface Functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\*

addFrontList

param: lst the linkedList

param: e the element to be added

pre: lst is not null

post: lst is not empty, increased size by 1

\*/

void addFrontList(struct linkedList \*lst, TYPE e)

{

\_addLink(lst, lst->firstLink->next, e);

}

/\*

addBackList

param: lst the linkedList

param: e the element to be added

pre: lst is not null

post: lst is not empty, increased size by 1

\*/

void addBackList(struct linkedList \*lst, TYPE e) {

\_addLink(lst, lst->lastLink, e);

}

/\*

frontList

param: lst the linkedList

pre: lst is not null

pre: lst is not empty

post: none

\*/

TYPE frontList (struct linkedList \*lst) {

assert (!EQ(lst, 0));

assert(!isEmptyList(lst));

return lst->firstLink->next->value;

}

/\*

backList

param: lst the linkedList

pre: lst is not null

pre: lst is not empty

post: lst is not empty

\*/

TYPE backList(struct linkedList \*lst)

{

assert (!EQ(lst, 0));

assert(!isEmptyList(lst));

return lst->lastLink->prev->value;

}

/\*

removeFrontList

param: lst the linkedList

pre:lst is not null

pre: lst is not empty

post: size is reduced by 1

\*/

void removeFrontList(struct linkedList \*lst) {

assert (!EQ(lst, 0));

assert(!isEmptyList(lst));

\_removeLink(lst,lst->firstLink->next);

}

/\*

removeBackList

param: lst the linkedList

pre: lst is not null

pre:lst is not empty

post: size reduced by 1

\*/

void removeBackList(struct linkedList \*lst)

{

assert (!EQ(lst, 0));

assert(!isEmptyList(lst));

\_removeLink(lst, lst->lastLink);

}

int Contains (struct linkedList \*lst, TYPE e)

{

assert (!EQ(lst, 0));

struct DLink\* temp = lst->firstLink->next;

while (!EQ(temp, lst->lastLink)){

if(EQ(temp->value, e)){

return 1;

} else {

temp = temp->next;

}

}

return 0;

}

/\* Iterative implementation of remove()

Function to remove a given value from the list

Pre: lst is not null

\*/

void listRemove (struct linkedList \*lst, TYPE e)

{

assert (!EQ(lst, 0));

struct DLink \*temp = lst->firstLink->next;

if(Contains(lst, e)){

while (!EQ(lst->lastLink, temp)){

if(EQ(temp->value, e)){

\_removeLink(lst, temp);

break;

} else {

temp = temp->next;

}

}

}

}

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Bag Interface Functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\*

Add an item to the bag

param: lst pointer to the bag

param: v value to be added

pre: lst is not null

post: a link storing val is added to the bag

\*/

void addList(struct linkedList \*lst, TYPE v)

{

\_addLinkBefore(lst, lst->firstLink->next, v);

}

/\* Returns boolean (encoded as an int) demonstrating whether or not

the specified value is in the collection

true = 1

false = 0

param: lst pointer to the bag

param: e the value to look for in the bag

pre: lst is not null

pre: lst is not empty

post: no changes to the bag

\*/

int containsList (struct linkedList \*lst, TYPE e) {

assert (!EQ(lst, 0));

struct DLink\* temp = lst->firstLink->next;

while (!EQ(temp, lst->lastLink)){

if(EQ(temp->value, e)){

return 1;

} else {

temp = temp->next;

}

}

return 0;

}

/\* Removes the first occurrence of the specified value from the collection

if it occurs

param: lst pointer to the bag

param: e the value to be removed from the bag

pre: lst is not null

pre: lst is not empty

post: e has been removed

post: size of the bag is reduced by 1

\*/

void removeList (struct linkedList \*lst, TYPE e) {

assert (!EQ(lst, 0));

struct DLink \*temp = lst->firstLink->next;

if(containsList(lst, e)){

while (!EQ(lst->lastLink, temp)){

if(EQ(temp->value, e)){

\_removeLink(lst, temp);

break;

} else {

temp = temp->next;

}

}

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**/\*** linkedList**.h: LinkedList Header with function declaration. \*/**

#ifndef \_\_LISTDEQUE\_H

#define \_\_LISTDEQUE\_H

# ifndef TYPE

# define TYPE int

# define TYPE\_SIZE sizeof(int)

# endif

# ifndef LT

# define LT(A, B) ((A) < (B))

# endif

# ifndef EQ

# define EQ(A, B) ((A) == (B))

# endif

struct linkedList;

struct linkedList \*createLinkedList();

void deleteLinkedList(struct linkedList \*lst);

/\* Deque Interface \*/

int isEmptyList(struct linkedList \*lst);

void addBackList(struct linkedList \*lst, TYPE e);

void addFrontList(struct linkedList \*lst, TYPE e);

TYPE frontList(struct linkedList \*lst);

TYPE backList(struct linkedList \*lst);

void removeFrontList(struct linkedList \*lst);

void removeBackList(struct linkedList \*lst);

/\*Bag Interface \*/

void addList(struct linkedList \*lst, TYPE v);

int containsList(struct linkedList \*lst, TYPE e);

void removeList(struct linkedList \*lst, TYPE e);

#endif

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**/\*** linkedListMain.c **: LinkedList main function . \*/**

#include "linkedList.h"

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

/\* VISUAL STUDIO (VS) USERS: COMMENT OUT THE LINE BELOW

TO EXCLUDE THE MEMORY TEST CODE.

\*/

// #define MEMORY\_TEST\_INCLUDED

// #ifdef MEMORY\_TEST\_INCLUDED

/\* This header is needed for memory usage calculation. \*/

// #include <sys/resource.h>

// \*/

/\* Function to get current memory usage in KB (Max Resident Set Size)

long getMemoryUsage() {

int who = RUSAGE\_SELF;

struct rusage usage;

int ret;

ret = getrusage(who, &usage);

return usage.ru\_maxrss;

}

#endif

\*/

/\*Function to get number of milliseconds elapsed since program started execution\*/

double getMilliseconds() {

return 1000.0 \* clock() / CLOCKS\_PER\_SEC;

}

int main(int argc, char\* argv[]){

struct linkedList\* b;

int n, i;

double t1, t2;

#ifdef MEMORY\_TEST\_INCLUDED

/\* variables to hold memory used before and after creating LinkedList \*/

long m1, m2;

/\* memory used BEFORE creating LinkedList \*/

m1 = getMemoryUsage();

#endif

if( argc != 2 ) return 0;

b = createLinkedList();

n = atoi(argv[1]);/\*number of elements to add\*/

for( i = 0 ; i < n; i++) {

addList(b, (TYPE)i);/\*Add elements\*/

}

#ifdef MEMORY\_TEST\_INCLUDED

/\* memory used AFTER creating LinkedList \*/

m2 = getMemoryUsage();

printf("Memory used by LinkedList: %ld KB \n", m2-m1);

#endif

t1 = getMilliseconds();/\*Time before contains()\*/

for(i=0; i<n; i++) {

containsList(b, i);

}

t2 = getMilliseconds();/\*Time after contains()\*/

printf("Time for running contains() on %d elements: %g ms\n", n, t2-t1);

/\* delete the linked list \*/

deleteLinkedList(b);

return 0;

}

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**/\*** makefilelinkedList.txt **: Make file if Manual compiling is done . \*/**

default: prog

linkedList.o: linkedList.c linkedList.h

gcc -Wall -std=c99 -c linkedList.c

prog: linkedList.o

gcc -Wall -std=c99 -o prog linkedList.o linkedListMain.c

clean:

rm linkedList.o

cleanall: clean

rm prog