**DEQUE USING CIRCULARLY-DOUBLY-LINKED LIST in C**

**/\* cirListDeque.c \*/**

#include <stdio.h>

#include <stdlib.h>

#include <assert.h>

#include <float.h>

#include "cirListDeque.h"

/\* Double Link Struture \*/

struct DLink {

TYPE value;/\* value of the link \*/

struct DLink \* next;/\* pointer to the next link \*/

struct DLink \* prev;/\* pointer to the previous link \*/

};

# define TYPE\_SENTINEL\_VALUE DBL\_MAX

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

Deque ADT based on Circularly-Doubly-Linked List WITH Sentinel

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

struct cirListDeque {

int size;/\* number of links in the deque \*/

struct DLink \*Sentinel; /\* pointer to the sentinel \*/

};

/\* internal functions prototypes \*/

struct DLink\* \_createLink (TYPE val);

void \_addLinkAfter(struct cirListDeque \*q, struct DLink \*lnk, TYPE v);

void \_removeLink(struct cirListDeque \*q, struct DLink \*lnk);

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

Deque Functions

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

/\* Initialize deque.

param: q pointer to the deque

pre: q is not null

post: q->Sentinel is allocated and q->size equals zero

\*/

void \_initCirListDeque (struct cirListDeque \*q)

{

assert (!EQ(q, 0));

q->Sentinel = (struct DLink \*)malloc(sizeof(struct DLink));

assert (!EQ(q->Sentinel, 0));

q->Sentinel->next = q->Sentinel;

q->Sentinel->prev = q->Sentinel;

q->size = 0;

}

/\*

create a new circular list deque

\*/

struct cirListDeque \*createCirListDeque()

{

struct cirListDeque \*newCL = malloc(sizeof(struct cirListDeque));

\_initCirListDeque(newCL);

return(newCL);

}

/\* Create a link for a value.

param: val the value to create a link for

pre: none

post: a link to store the value

\*/

struct DLink \* \_createLink (TYPE val)

{

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

assert (!EQ(newLink, 0));

newLink->value = val;

return newLink;

}

/\* Adds a link after another link

param: q pointer to the deque

param: lnk pointer to the existing link in the deque

param: v value of the new link to be added after the existing link

pre: q is not null

pre: lnk is not null

pre: lnk is in the deque

post: the new link is added into the deque after the existing link

\*/

void \_addLinkAfter(struct cirListDeque \*q, struct DLink \*lnk, TYPE v)

{

assert (!EQ(q, 0));

assert (!EQ(lnk, 0));

struct DLink \*newLink = \_createLink(v);

assert (!EQ(newLink, 0));

lnk->next->prev = newLink; //right connects to new

newLink->next = lnk->next; //new connect to right

newLink->prev = lnk; //new connect to left

lnk->next = newLink; //left connect to new

q->size++;

}

/\* Adds a link to the back of the deque

param: q pointer to the deque

param: val value for the link to be added

pre: q is not null

post: a link storing val is added to the back of the deque

\*/

void addBackCirListDeque (struct cirListDeque \*q, TYPE val)

{

assert (!EQ(q, 0));

\_addLinkAfter(q, q->Sentinel->prev, val);

}

/\* Adds a link to the front of the deque

param: q pointer to the deque

param: val value for the link to be added

pre: q is not null

post: a link storing val is added to the front of the deque

\*/

void addFrontCirListDeque(struct cirListDeque \*q, TYPE val)

{

assert (!EQ(q, 0));

\_addLinkAfter(q, q->Sentinel, val);

}

/\* Get the value of the front of the deque

param: q pointer to the deque

pre: q is not null and q is not empty

post: none

ret: value of the front of the deque

\*/

TYPE frontCirListDeque(struct cirListDeque \*q)

{

assert (!EQ(q, 0));

assert (!EQ(q->size, 0));

return q->Sentinel->next->value;

}

/\* Get the value of the back of the deque

param: q pointer to the deque

pre: q is not null and q is not empty

post: none

ret: value of the back of the deque

\*/

TYPE backCirListDeque(struct cirListDeque \*q)

{

assert (!EQ(q, 0));

assert (!EQ(q->size, 0));

return q->Sentinel->prev->value;

}

/\* Remove a link from the deque

param: q pointer to the deque

param: lnk pointer to the link to be removed

pre: q is not null and q is not empty

post: the link is removed from the deque

\*/

void \_removeLink(struct cirListDeque \*q, struct DLink \*lnk)

{

assert (!EQ(q, 0));

struct DLink \*temp = q->Sentinel->next;

while (!EQ(temp, q->Sentinel)){

if(EQ(temp, lnk)){

lnk->prev->next = lnk->next;

lnk->next->prev = lnk->prev;

free(lnk);

q->size--;

break;

} else {

temp = temp->next;

}

}

}

/\* Remove the front of the deque

param: q pointer to the deque

pre: q is not null and q is not empty

post: the front is removed from the deque

\*/

void removeFrontCirListDeque (struct cirListDeque \*q) {

assert (!EQ(q, 0));

assert (!isEmptyCirListDeque(q));

\_removeLink(q, q->Sentinel->next);

}

/\* Remove the back of the deque

param: q pointer to the deque

pre: q is not null and q is not empty

post: the back is removed from the deque

\*/

void removeBackCirListDeque(struct cirListDeque \*q)

{

assert (!EQ(q, 0));

\_removeLink(q, q->Sentinel->prev);

}

/\* De-allocate all links of the deque

param: q pointer to the deque

pre: none

post: All links (including Sentinel) are de-allocated

\*/

void freeCirListDeque(struct cirListDeque \*q)

{

while (!EQ(q->Sentinel->next, q->Sentinel)){

removeBackCirListDeque(q);

}

free(q->Sentinel);

free(q);

}

/\* Deallocate all the links and the deque itself.

param: v pointer to the dynamic array

pre: v is not null

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

\*/

void deleteCirListDeque(struct cirListDeque \*q) {

assert(q != 0);

freeCirListDeque(q);

free(q);

}

/\* Check whether the deque is empty

param: q pointer to the deque

pre: q is not null

ret: 1 if the deque is empty. Otherwise, 0.

\*/

int isEmptyCirListDeque(struct cirListDeque \*q)

{

assert (!EQ(q, 0));

return (EQ(0, q->size));

}

/\* Print the links in the deque from front to back

param: q pointer to the deque

pre: q is not null and q is not empty

post: the links in the deque are printed from front to back

\*/

void printCirListDeque(struct cirListDeque \*q)

{

assert (!EQ(q, 0));

assert (!isEmptyCirListDeque(q));

int i;

struct DLink \*temp = q->Sentinel->next;

printf("\n");

for (i = 0; i < q->size; i++){

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

temp = temp->next;

}

}

/\* Reverse the deque

param: q pointer to the deque

pre: q is not null and q is not empty

post: the deque is reversed

\*/

void reverseCirListDeque(struct cirListDeque \*q)

{

assert (!EQ(q, 0));

struct cirListDeque\* newQ = createCirListDeque();

while(!isEmptyCirListDeque(q)){

addBackCirListDeque(newQ, backCirListDeque(q));

removeBackCirListDeque(q);

}

q->Sentinel = newQ->Sentinel;

q->size = newQ->size;

free(newQ);

}  
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++c

**/\*cirListDeque.h \*/**

#ifndef \_\_CIRLISTDEQUE\_H

#define \_\_CIRLISTDEQUE\_H

# ifndef TYPE

# define TYPE double

# define TYPE\_SIZE sizeof(double)

# endif

# ifndef LT

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

# endif

# ifndef EQ

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

# endif

/\* struct prototype \*/

struct cirListDeque;

struct cirListDeque \*createCirListDeque();

void deleteCirListDeque(struct cirListDeque \*q);

int isEmptyCirListDeque(struct cirListDeque \*q);

void addBackCirListDeque(struct cirListDeque \*q, TYPE val);

void addFrontCirListDeque(struct cirListDeque \*q, TYPE val);

TYPE frontCirListDeque(struct cirListDeque \*q);

TYPE backCirListDeque(struct cirListDeque \*q);

void removeFrontCirListDeque(struct cirListDeque \*q);

void removeBackCirListDeque(struct cirListDeque \*q);

void freeCirListDeque(struct cirListDeque \*q);

void printCirListDeque(struct cirListDeque \*q);

void reverseCirListDeque(struct cirListDeque \*q);

#endif

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++c

**/\*listDequeTest.c\*/**

#include "cirListDeque.h"

#include <stdio.h>

int main(){

struct cirListDeque\* q = createCirListDeque();

printf("---ADDING 1,2,3 TO BACK----\n");

addBackCirListDeque(q, (TYPE)1);

addBackCirListDeque(q, (TYPE)2);

addBackCirListDeque(q, (TYPE)3);

printf("---Printing DEQUE after Adding 1,2,3----\n");

printCirListDeque(q);

printf("---ADDING 4,5,6 TO FRONT----\n");

addFrontCirListDeque(q, (TYPE)4);

addFrontCirListDeque(q, (TYPE)5);

addFrontCirListDeque(q, (TYPE)6);

printf("---Printing DEQUE after Adding 4,5,6----\n");

printCirListDeque(q);

printf("---Printing FULL DEQUE----\n");

printCirListDeque(q);

printf("---Printing Head----\n");

printf("%g\n", frontCirListDeque(q));

printf("---Printing Tail----\n");

printf("%g\n", backCirListDeque(q));

removeFrontCirListDeque(q);

removeBackCirListDeque(q);

printf("---Deque after removing Head and Tail----\n");

printCirListDeque(q);

printf("---Reversing Deque ----\n");

reverseCirListDeque(q);

printf("--- Deque after Reversing----\n");

printCirListDeque(q);

printf("---Deleting Deque ----\n");

deleteCirListDeque(q);

return 0;

}

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

**/\* makefilecirListDeque.txt\*/**

default: prog

cirListDeque.o: cirListDeque.c cirListDeque.h

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

prog: cirListDeque.o

gcc -Wall -std=c99 -o prog cirListDeque.o listDequeTest.c

clean:

rm cirListDeque.o

cleanall: clean

rm prog

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++c