**DEQUE IMPLEMENTATION USING DYNAMIC ARRAY AND STRUCTURE IN C**

**/\* dynamicArrayDeque.c \*/**

#include <assert.h>

#include <stdlib.h>

#include <stdio.h>

# ifndef TYPE

# define TYPE double

# define TYPE\_SIZE sizeof(double)

# endif

struct deque {

TYPE \*data; /\* Pointer to data array. \*/

int size; /\* Number of elements in collection. \*/

int beg; /\* Index of first element. \*/

int capacity; /\* Capacity of array. \*/

};

void \_dequeSetCapacity (struct deque \*v, int newCap) {

int i;

/\* Create a new underlying array\*/

TYPE \*newData = (TYPE\*)malloc(sizeof(TYPE)\*newCap);

assert(newData != 0);

/\* copy elements to it \*/

int j = v->beg;

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

{

newData[i] = v->data[j];

j = j + 1;

if(j >= v->capacity)

j = 0;

}

/\* Delete the oldunderlying array\*/

free(v->data);

/\* update capacity and size and data\*/

v->data = newData;

v->capacity = newCap;

v->beg = 0;

}

void dequeFree (struct deque \*d) {

free(d->data); d->size = 0; d->capacity = 0;

}

void dequeInit (struct deque \*d, int initCapacity) {

d->size = d->beg = 0;

d->capacity = initCapacity; assert(initCapacity > 0);

d->data = (TYPE \*) malloc(initCapacity \* sizeof(TYPE));

assert(d->data != 0);

}

int dequeSize (struct deque \*d) { return d->size; }

void \_dequeDoubleCapacity (struct deque \*d);

void dequeAddFront (struct deque \*d, TYPE newValue) {

assert(d != 0);

if (d->size >= d->capacity) \_dequeSetCapacity(d, 2\*d->capacity);

//check if the beginning is zero

if (d->beg == 0)

{

d->data[d->capacity - 1] = newValue; //since the beginning is zero, need to add to the end

d->beg = d->capacity-1; //adjust d->beg to be the end of the array

} else //beginning is not 0

{

d->data[d->beg-1] = newValue; //add one space in front of beginning

d->beg--; //decrement beginning

}

d->size++; //increment the size

}

void dequeAddBack (struct deque \*d, TYPE newValue) {

assert(d != 0);

if (d->size >= d->capacity) \_dequeSetCapacity(d, 2\* d->capacity);

int index = d->beg + d->size;

/\* Check if the deque has wrapped around \*/

if (index > d->capacity)

{

index -= d->capacity; //if it has, substract the capacity to find the logical end

}

d->data[index] = newValue;

d->size++;

}

TYPE dequeFront (struct deque \*d) {

assert(d != 0);

return d->data[d->beg];

}

TYPE dequeBack (struct deque \*d) {

assert(d != 0);

int index = d->beg + d->size - 1;

/\* Check if the deque has wrapped around \*/

if (index > d->capacity)

{

index -= d->capacity; //subtract capacity to find the logical end

}

return d->data[index];

}

void dequeRemoveFront (struct deque \*d) {

assert(d != 0);

assert(d->size > 0); //make sure the deque isn’t empty

/\* If the beginning is the last physical index, needs to be reset to zero \*/

if (d->beg == d->capacity-1)

{

d->beg = 0;

} else //otherwise, just bump the beginning up

{

d->beg++;

}

d->size--; //decrement the size

}

void dequeRemoveBack (struct deque \*d) {

assert(d != 0);

assert(d->size > 0); //make sure the deque isn’t empty

d->size--; //decrement the size

}

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

struct deque d;

//initDynArrDeque(&d, 5);

dequeInit(&d, 5); // cap =5,cnt/index = 0,beg=0

//addBackArrDeque(&d, 3.0);

// int cnt = (int)d->data[index];

dequeAddBack(&d, 3.0); //cap =5,cnt/index = 1,beg=0

//addBackArrDeque(&d, 5.0);

dequeAddBack(&d, 3.0); //cap =5,cnt/index = 1,beg=0

//addBackArrDeque(&d, 1.0);

dequeAddBack(&d, 1.0); //cap =5,cnt/index = 2,beg=0

//removeFrontArrDeque(&d);

dequeRemoveFront(&d); //cap =5,cnt/size = 2,beg=1

//addBackArrDeque(&d, 2.0);

dequeAddBack(&d, 2.0); //cap =5,cnt/size = 3,beg=1

//addFrontArrDeque(&d, 10.0);

dequeAddFront(&d, 10.0); //cap =5,cnt/size = 4,beg=0

//removeBackArrDeque(&d);//cap =5,cnt/size = 3,beg=0

dequeRemoveBack(&d);//cap =5,cnt/size = 2,beg=1

//removeFront(ArrDeque &d);

dequeRemoveFront(&d);

return 0;

}

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