***// DOUBLE ENDED QUEUE***

#include <stdlib.h>

#include <stdio.h>

typedef struct node node;

struct node

{

int data;

node \*link;

};

typedef struct list

{

node \*head;

node \*tail;

int number\_of\_nodes;

}s\_list;

typedef struct queue

{

s\_list \*ptr\_list;

}queue;

void list\_initialize(s\_list\* list); *// Initializes the head, tail and no of nodes of the list.*

void list\_insert\_rear(s\_list\* list, int data); *// Inserts element at the rear of the list.*

void list\_insert\_front(s\_list\* list, int data); *// Inserts element at the front of the list.*

void list\_delete\_front(s\_list \*list); *// Deletes element at the front of the list. No operation if list is empty.*

void list\_delete\_rear(s\_list\* list); *// Deletes element at the rear of the list. No operation if list is empty.*

const int node\_get\_data(node\* node\_ptr);

void list\_destroy(s\_list\* list);

void display(s\_list\* list); *// Prints the list. Prints "Empty Queue" if list is empty.*

void queue\_initialize(queue\* ptr\_queue);

void queue\_enqueue\_rear(queue\* ptr\_queue, int data);

void queue\_dequeue\_front(queue\* ptr\_queue);

void queue\_enqueue\_front(queue\* ptr\_queue, int data);

void queue\_dequeue\_rear(queue\* ptr\_queue);

int queue\_is\_empty(queue\* ptr\_queue); *// Returns a boolean value.*

const int queue\_peek(queue\* ptr\_queue); *// Returns the element at the front.*

void queue\_destroy(queue\* ptr\_queue);

void print\_queue(queue\* ptr\_queue);

int main()

{

queue queue;

queue\_initialize(&queue);

int choice,element;

int loop = 1;

while(loop)

{

scanf("%d", &choice);

switch(choice)

{

case 0:

scanf("%d", &element);

queue\_enqueue\_front(&queue, element);

break;

case 1:

scanf("%d", &element);

queue\_enqueue\_rear(&queue, element);

break;

case 2:

queue\_dequeue\_front(&queue);

break;

case 3:

queue\_dequeue\_rear(&queue);

break;

case 4:

if(queue\_is\_empty(&queue))

{

printf("Empty Queue\n");

}

else

{

printf("%d\n", queue\_peek(&queue));

}

break;

case 5:

print\_queue(&queue);

break;

default:

queue\_destroy(&queue);

loop = 0;

}

}

return 0;

}

static node\* create\_node(int data, node\* link)

{

link = (node\*)malloc(sizeof(node));

link->data = data;

link->link = NULL;

return link;

}

void list\_initialize(s\_list\* ptr\_list)

{

ptr\_list->head = NULL;

ptr\_list->tail = NULL;

ptr\_list->number\_of\_nodes = 0;

}

void list\_insert\_front(s\_list\* ptr\_list, int data)

{

node \*temp = create\_node(data,temp);

if(ptr\_list->head == NULL)

{

ptr\_list->head = ptr\_list->tail = temp;

}

else

{

temp->link = ptr\_list->head;

ptr\_list->head = temp;

}

ptr\_list->number\_of\_nodes++;

}

void list\_insert\_rear(s\_list\* ptr\_list, int data)

{

node\* temp = create\_node(data,temp);

if(ptr\_list->head ==NULL)

{

ptr\_list->head = ptr\_list->tail = temp;

}

else

{

ptr\_list->tail->link = temp;

ptr\_list->tail = temp;

}

ptr\_list->number\_of\_nodes++;

}

const int node\_get\_data(node \*node\_ptr)

{

return node\_ptr->data;

}

void list\_delete\_front(s\_list\* ptr\_list)

{

if(ptr\_list->head == NULL) return;

else if(ptr\_list->head==ptr\_list->tail) *//ONLY ONE NODE*

{

node \*temp = ptr\_list->head;

ptr\_list->tail = ptr\_list->head = NULL;

free(temp);

ptr\_list->number\_of\_nodes--;

}

else

{

node \*temp = ptr\_list->head;

ptr\_list->head = ptr\_list->head->link;

free(temp);

ptr\_list->number\_of\_nodes--;

}

}

void list\_delete\_rear(s\_list\* ptr\_list)

{

if(ptr\_list->tail==NULL)

{

return;

}

else if(ptr\_list->head==ptr\_list->tail)

{

node \*temp = ptr\_list->head;

ptr\_list->tail = ptr\_list->head = NULL;

free(temp);

ptr\_list->number\_of\_nodes--;

}

else

{

node \*temp = ptr\_list->head, \*prev = NULL;

while(temp->link!=NULL)

{

prev = temp;

temp=temp->link;

}

prev->link = NULL;

ptr\_list->tail = prev;

free(temp);

ptr\_list->number\_of\_nodes--;

}

}

void display(s\_list\* ptr\_list)

{

if(ptr\_list->head==NULL)

{

printf("Empty Queue");

}

else

{

node\* ptr\_h = ptr\_list->head;

while(ptr\_h!=NULL)

{

printf("%d ",ptr\_h->data);

ptr\_h = ptr\_h->link;

}

}

printf("\n");

}

void list\_destroy(s\_list\* ptr\_list)

{

node\* q = ptr\_list->head;

node\* r;

while(q!=NULL)

{

r = q;

q = q->link;

free(r);

}

free(ptr\_list);

}

int queue\_is\_empty(queue\* ptr)

{

if(ptr->ptr\_list->head==NULL)

return 1;

else return 0;

}

const int queue\_peek(queue\* ptr)

{

return ptr->ptr\_list->head->data;

}

void queue\_initialize(queue\* queue\_list)

{

queue\_list -> ptr\_list = (s\_list\*)malloc(sizeof(s\_list));

list\_initialize(queue\_list->ptr\_list);

}

void queue\_enqueue\_rear(queue\* ptr,int data)

{

list\_insert\_rear(ptr->ptr\_list,data);

}

void queue\_dequeue\_front(queue\* ptr)

{

list\_delete\_front(ptr->ptr\_list);

}

void queue\_destroy(queue\* ptr)

{

list\_destroy(ptr->ptr\_list);

}

void queue\_enqueue\_front(queue\* ptr, int data)

{

list\_insert\_front(ptr->ptr\_list,data);

}

void queue\_dequeue\_rear(queue\* ptr)

{

list\_delete\_rear(ptr->ptr\_list);

}

void print\_queue(queue\* ptr)

{

display(ptr->ptr\_list);

}