#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

// Queue capacity

#define CAPACITY 100

/\*\*

\* Global queue declaration.

\*/

int queue[CAPACITY];

unsigned int size = 0;

unsigned int rear = CAPACITY - 1; // Initally assumed that rear is at end

unsigned int front = 0;

/\* Function declaration for various operations on queue \*/

int enqueue(int data);

int dequeue();

int isFull();

int isEmpty();

int getRear();

int getFront();

/\* Driver function \*/

int main()

{

int ch, data;

/\* Run indefinitely until user manually terminates \*/

while (1)

{

/\* Queue menu \*/

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

printf(" QUEUE ARRAY IMPLEMENTATION PROGRAM \n");

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

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Size\n");

printf("4. Get Rear\n");

printf("5. Get Front\n");

printf("0. Exit\n");

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

printf("Select an option: ");

scanf("%d", &ch);

/\* Menu control switch \*/

switch (ch)

{

case 1:

printf("\nEnter data to enqueue: ");

scanf("%d", &data);

// Enqueue function returns 1 on success

// otherwise 0

if (enqueue(data))

printf("Element added to queue.");

else

printf("Queue is full.");

break;

case 2:

data = dequeue();

// on success dequeue returns element removed

// otherwise returns INT\_MIN

if (data == INT\_MIN)

printf("Queue is empty.");

else

printf("Data => %d", data);

break;

case 3:

// isEmpty() function returns 1 if queue is emtpy

// otherwise returns 0

if (isEmpty())

printf("Queue is empty.");

else

printf("Queue size => %d", size);

break;

case 4:

if (isEmpty())

printf("Queue is empty.");

else

printf("Rear => %d", getRear());

break;

case 5:

if (isEmpty())

printf("Queue is empty.");

else

printf("Front => %d", getFront());

break;

case 0:

printf("Exiting from app.\n");

exit(0);

default:

printf("Invalid choice, please input number between (0-5).");

break;

}

printf("\n\n");

}

}

/\*\*

\* Enqueue/Insert an element to the queue.

\*/

int enqueue(int data)

{

// Queue is full throw Queue out of capacity error.

if (isFull())

{

return 0;

}

// Ensure rear never crosses array bounds

rear = (rear + 1) % CAPACITY;

// Increment queue size

size++;

// Enqueue new element to queue

queue[rear] = data;

// Successfully enqueued element to queue

return 1;

}

/\*\*

\* Dequeue/Remove an element from the queue.

\*/

int dequeue()

{

int data = INT\_MIN;

// Queue is empty, throw Queue underflow error

if (isEmpty())

{

return INT\_MIN;

}

// Dequeue element from queue

data = queue[front];

// Ensure front never crosses array bounds

front = (front + 1) % CAPACITY;

// Decrease queue size

size--;

return data;

}

/\*\*

\* Checks if queue is full or not. It returns 1 if queue is full,

\* overwise returns 0.

\*/

int isFull()

{

return (size == CAPACITY);

}

/\*\*

\* Checks if queue is empty or not. It returns 1 if queue is empty,

\* otherwise returns 0.

\*/

int isEmpty()

{

return (size == 0);

}

/\*\*

\* Gets, front of the queue. If queue is empty return INT\_MAX otherwise

\* returns front of queue.

\*/

int getFront()

{

return (isEmpty())

? INT\_MIN

: queue[front];

}

/\*\*

\* Gets, rear of the queue. If queue is empty return INT\_MAX otherwise

\* returns rear of queue.

\*/

int getRear()

{

return (isEmpty())

? INT\_MIN

: queue[rear];

}

