**Queue & Circular Queue**

Problem Statement:

Write C program to implement a  Queue  and a Circular Queue using array. Consider 1000 enqueue and dequeue operations at random (Generate a random number 0 or 1 to identify enqueue or dequeue operation). Report the number of Stack full and Stack empty conditions in each case after every 25 operations.

• Output example :

For Normal Queue--->

For 1 th case:

Full: 0

Empty: 5

For 2 th case:

Full: 1

Empty: 7

For 3 th case:

Full: 0

Empty: 0

For 4 th case:

Full: 1

Empty: 7

For 5 th case:

Full: 4

Empty: 0

For 6 th case:

Full: 0

Empty: 2

For 7 th case:

Full: 4

Empty: 0

For 8 th case:

Full: 0

Empty: 5

For 9 th case:

Full: 0

Empty: 7

For 10 th case:

Full: 4

Empty: 4

For 11 th case:

Full: 0

Empty: 2

For 12 th case:

Full: 1

Empty: 3

For 13 th case:

Full: 1

Empty: 2

For 14 th case:

Full: 0

Empty: 4

For 15 th case:

Full: 5

Empty: 0

For 16 th case:

Full: 0

Empty: 9

For 17 th case:

Full: 0

Empty: 4

For 18 th case:

Full: 0

Empty: 0

For 19 th case:

Full: 0

Empty: 6

For 20 th case:

Full: 0

Empty: 3

For 21 th case:

Full: 0

Empty: 4

For 22 th case:

Full: 1

Empty: 2

For 23 th case:

Full: 0

Empty: 1

For 24 th case:

Full: 3

Empty: 0

For 25 th case:

Full: 0

Empty: 7

For 26 th case:

Full: 0

Empty: 3

For 27 th case:

Full: 1

Empty: 4

For 28 th case:

Full: 4

Empty: 0

For 29 th case:

Full: 4

Empty: 0

For 30 th case:

Full: 5

Empty: 0

For 31 th case:

Full: 3

Empty: 0

For 32 th case:

Full: 0

Empty: 10

For 33 th case:

Full: 3

Empty: 0

For 34 th case:

Full: 8

Empty: 0

For 35 th case:

Full: 0

Empty: 2

For 36 th case:

Full: 3

Empty: 3

For 37 th case:

Full: 1

Empty: 1

For 38 th case:

Full: 0

Empty: 6

For 39 th case:

Full: 0

Empty: 3

For 40 th case:

Full: 3

Empty: 0

For Circular Queue--->

For 1 th case:

Full: 1

Empty: 7

For 2 th case:

Full: 0

Empty: 10

For 3 th case:

Full: 0

Empty: 6

For 4 th case:

Full: 1

Empty: 3

For 5 th case:

Full: 1

Empty: 9

For 6 th case:

Full: 6

Empty: 1

For 7 th case:

Full: 4

Empty: 7

For 8 th case:

Full: 3

Empty: 11

For 9 th case:

Full: 8

Empty: 8

For 10 th case:

Full: 8

Empty: 2

For 11 th case:

Full: 0

Empty: 8

For 12 th case:

Full: 0

Empty: 7

For 13 th case:

Full: 0

Empty: 12

For 14 th case:

Full: 1

Empty: 3

For 15 th case:

Full: 5

Empty: 3

For 16 th case:

Full: 0

Empty: 10

For 17 th case:

Full: 0

Empty: 5

For 18 th case:

Full: 4

Empty: 6

For 19 th case:

Full: 4

Empty: 4

For 20 th case:

Full: 2

Empty: 4

For 21 th case:

Full: 3

Empty: 2

For 22 th case:

Full: 0

Empty: 9

For 23 th case:

Full: 1

Empty: 7

For 24 th case:

Full: 4

Empty: 9

For 25 th case:

Full: 3

Empty: 2

For 26 th case:

Full: 4

Empty: 6

For 27 th case:

Full: 3

Empty: 4

For 28 th case:

Full: 2

Empty: 2

For 29 th case:

Full: 0

Empty: 2

For 30 th case:

Full: 3

Empty: 6

For 31 th case:

Full: 3

Empty: 7

For 32 th case:

Full: 9

Empty: 3

For 33 th case:

Full: 6

Empty: 3

For 34 th case:

Full: 4

Empty: 5

For 35 th case:

Full: 2

Empty: 1

For 36 th case:

Full: 5

Empty: 1

For 37 th case:

Full: 5

Empty: 4

For 38 th case:

Full: 0

Empty: 12

For 39 th case:

Full: 3

Empty: 11

For 40 th case:

Full: 5

Empty: 2

Proposed C Code:

/\* ------- main.c ------- \*/

#include <stdio.h>

#include <stdlib.h>

// Normal Queue

int rear, front;

int \*queue;

int size = 5;

int f = -1, e = -1;

void init()

{

rear = -1;

front = -1;

f = 0;

e = 0;

queue = (int \*)calloc(100, sizeof(int));

}

void enqueue(int item)

{

if (rear - front == size)

{

f++;

return;

}

queue[++rear] = item;

}

int dequeue()

{

if (front == rear)

{

e++;

return -1;

}

return queue[++front];

}

// Circular Queue

int cirrear, cirfront;

int \*cirqueue;

int cirsize = 3;

int cf = -1, ce = -1;

void cir\_init()

{

cirrear = -1;

cirfront = -1;

cf = 0;

ce = 0;

cirqueue = (int \*)calloc(cirsize, sizeof(int));

}

void cir\_enqueue(int item)

{

if (cirfront == (cirrear + 1) % cirsize)

{

cf++;

return;

}

if (cirfront == -1)

{

cirfront = 0;

}

cirrear = (cirrear + 1) % cirsize;

cirqueue[cirrear] = item;

return;

}

int cir\_dequeue()

{

if (cirfront == cirrear)

{

ce++;

return -1;

}

cirfront = (cirfront + 1) % cirsize;

return cirqueue[cirfront];

}

int main()

{

// For Normal Queue -->

printf("For Normal Queue--->\n");

for (int i = 0; i < 40; i++)

{

init();

for (int j = 0; j < 25; j++)

{

int x = rand() % 2;

if (x == 0)

{

enqueue(rand());

}

else

{

dequeue();

}

}

printf("For %d th case: \n", i + 1);

printf("Full: %d\n", f);

printf("Empty: %d\n\n", e);

}

// For Circular Queue-->

printf("For Circular Queue--->\n");

for (int i = 0; i < 40; i++)

{

cir\_init();

for (int j = 0; j < 25; j++)

{

int x = rand() % 2;

if (x == 0)

{

cir\_enqueue(rand());

}

else

{

cir\_dequeue();

}

}

printf("For %d th case: \n", i + 1);

printf("Full: %d\n", cf);

printf("Empty: %d\n\n", ce);

}

return 0;

}/\* ---------------------- \*/

Conclusion:

The proposed algorithm has a runtime of O(m\*n), where m=40 and n=25.

Limitations and assumptions for this algorithm include:

1. Here the size of both stack arrays is not expandable, it’s have a fix size.
2. For the normal queue we just moving the front pointer so it’s causing wastage of space.