

## แปลหมายเหตุ

```
#include <stdio.h>
#include <stdlib.h> // required for malloc()
// Queue ADT Type Definitions การกำหนดชนิดของคิว

typedef struct node
{
    void*      dataPtr;
    struct node* next;
} QUEUE_NODE;   นำคิวโหนดชี้ไปยัง Ptr
typedef struct
{
    QUEUE_NODE* front;
    QUEUE_NODE* rear;
    int      count;
} QUEUE;

// Prototype Declarations ประกาศตัวแปร
QUEUE* createQueue (void);
QUEUE* destroyQueue (QUEUE* queue);

bool dequeue (QUEUE* queue, void** itemPtr); // * = pointer พอยเตอร์
bool enqueue (QUEUE* queue, void* itemPtr); // ** = pointer of pointer
พอยเตอร์ชี้พอยเตอร์
bool queueFront (QUEUE* queue, void** itemPtr);
bool queueRear (QUEUE* queue, void** itemPtr);
int queueCount (QUEUE* queue);
bool emptyQueue (QUEUE* queue);
bool fullQueue (QUEUE* queue);
// End of Queue ADT Definitions จบการกำหนดชนิดของคิว

void printQueue (QUEUE* stack);

int main (void)
{
    // Local Definitions คำนิยามเฉพาะที่
    QUEUE* queue1;
    QUEUE* queue2;
    int* numPtr;
    int** itemPtr;

    // Statements คำสั่ง
    // Create two queues สร้างคิว 3 ตัว
    queue1 = createQueue();
```

```

queue2 = createQueue();
for (int i = 1; i <= 5; i++)
{
    numPtr = (int*)malloc(sizeof(i)); // set pointer to memory กำหนดตัวชี้
    ไปยังหน่วยความจำ
    *numPtr = i;
    enqueue(queue1, numPtr);
    if (!enqueue(queue2, numPtr))
    {
        printf ("\n\na**Queue overflow\n\n");
        exit (100);
    } // if !enqueue
} // for
printf ("Queue 1:\n");
printQueue (queue1); // 1 2 3 4 5
printf ("Queue 2:\n");
printQueue (queue2); // 1 2 3 4 5
return 0;
}

```

```

/* ===== createQueue ===== สร้างคิว =====
Allocates memory for a queue head node from dynamic
memory and returns its address to the caller.
Pre nothing
Post head has been allocated and initialized
Return head if successful; null if overflow
*/

```

```

QUEUE* createQueue (void)
{
    // Local Definitions คำนิยามเฉพาะที่
    QUEUE* queue;
    // Statements คำสั่ง
    queue = (QUEUE*) malloc (sizeof (QUEUE));
    if (queue)
    {
        queue->front = NULL;
        queue->rear = NULL;
        queue->count = 0;
    } // if
    return queue;
} // createQueue สร้างหัวของคิว

```

```

/* ===== enqueue ===== เพิ่มข้อมูลลงในคิว =====
This algorithm inserts data into a queue.

```

```

    Pre queue has been created
    Post data have been inserted
    Return true if successful, false if overflow
*/
bool enqueue (QUEUE* queue, void* itemPtr)
{
    // Local Definitions คำนิยามเฉพาะที่
    // QUEUE_NODE* newPtr; QUEUE_NODE เรียกใช้ newPtr
    // Statements คำสั่ง
    // if (!(newPtr = (QUEUE_NODE*)malloc(sizeof(QUEUE_NODE)))) return
    false;
        QUEUE_NODE* newPtr =
    (QUEUE_NODE*)malloc(sizeof(QUEUE_NODE));
        newPtr->dataPtr = itemPtr;
        newPtr->next = NULL;
        if (queue->count == 0)
            // Inserting into null queue แทรกลงในคิวว่าง
            queue->front = newPtr;
        else
            queue->rear->next = newPtr;
        (queue->count)++;
        queue->rear = newPtr;
        return true;
} // enqueue เพิ่มข้อมูลลงในคิว

/* ===== dequeue ===== ลบข้อมูลออกจากคิว =====
This algorithm deletes a node from the queue.
    Pre queue has been created
    Post Data pointer to queue front returned and
        front element deleted and recycled.
    Return true if successful; false if underflow
*/
bool dequeue (QUEUE* queue, void** itemPtr)
{
    // Local Definitions คำนิยามเฉพาะที่
    QUEUE_NODE* deleteLoc;
    // Statements คำสั่ง
    if (!queue->count)
        return false;
    *itemPtr = queue->front->dataPtr;
    deleteLoc = queue->front;
    if (queue->count == 1)
        // Deleting only item in queue ลบทุกอย่างออกจากคิว
        queue->rear = queue->front = NULL;

```

```

else
    queue->front = queue->front->next;
(queue->count)--;
free (deleteLoc);
return true;
} // dequeue   ลบข้อมูลออกจากคิว

/* ===== queueFront ===== การดึงข้อมูลตรงส่วนหัวออกมาใช้งาน=====
This algorithm retrieves data at front of the queue
queue without changing the queue contents.
Pre queue is pointer to an initialized queue
Post itemPtr passed back to caller
Return true if successful; false if underflow
*/
bool queueFront (QUEUE* queue, void** itemPtr)
{
// Statements คำสั่ง
if (!queue->count)
    return false;
else
    {
        *itemPtr = queue->front->dataPtr;
        return true;
    } // else
} // queueFront   ข้อมูลตรงส่วนหัวของคิวสามารถถูกเรียกหรือดึงขึ้นมาใช้ได้

/* ===== queueRear ===== การดึงข้อมูลตรงส่วนท้ายของคิวออกมาใช้งาน=====
Retrieves data at the rear of the queue
without changing the queue contents.
Pre queue is pointer to initialized queue
Post Data passed back to caller
Return true if successful; false if underflow
*/
bool queueRear (QUEUE* queue, void** itemPtr)
{
// Statements
if (!queue->count)
    return true;
else
    {
        *itemPtr = queue->rear->dataPtr;
        return false;
    } // else
} // queueRear   ดึงข้อมูลตรงส่วนท้ายของคิวมาใช้

```

```

/* ===== emptyQueue =====ทดสอบว่าคิวว่างหรือไม่=====
This algorithm checks to see if queue is empty
Pre queue is a pointer to a queue head node
Return true if empty; false if queue has data
*/
bool emptyQueue (QUEUE* queue)
{
// Statements
return (queue->count == 0);
} // emptyQueue คิวว่าง

/* ===== fullQueue =====ตรวจสอบว่าคิวเต็มหรือไม่=====
This algorithm checks to see if queue is full. It
is full if memory cannot be allocated for next node.
Pre queue is a pointer to a queue head node
Return true if full; false if room for a node
*/
bool fullQueue (QUEUE* queue)
{
// Check empty เช็คว่างหรือป่าว
if(emptyQueue(queue)) return false; // Not check in heap
// Local Definitions *
QUEUE_NODE* temp;
// Statements
temp = (QUEUE_NODE*)malloc(sizeof(*(queue->rear)));
if (temp)
{
free (temp);
return false; // Heap not full ไม่เต็ม
} // if
return true; // Heap full เต็ม
} // fullQueue ตรวจสอบว่าคิวเต็มหรือไม่

/* ===== queueCount =====จำนวนของคิว=====
Returns the number of elements in the queue.
Pre queue is pointer to the queue head node
Return queue count
*/
int queueCount(QUEUE* queue)
{
// Statements
return queue->count;
} // queueCount คืนค่าจำนวนสมาชิกล่าสุดที่อยู่ในคิวกลับไป

```

```

/* ===== destroyQueue =====ลบข้อมูลที่อยู่ในคิวออกทั้งหมด=====
Deletes all data from a queue and recycles its
memory, then deletes & recycles queue head pointer.
Pre Queue is a valid queue
Post All data have been deleted and recycled
Return null pointer
*/
QUEUE* destroyQueue (QUEUE* queue)
{
// Local Definitions คำนิยามเฉพาะที่
QUEUE_NODE* deletePtr;
// Statements คำสั่ง
if (queue)
{
while (queue->front != NULL)
{
free (queue->front->dataPtr);
deletePtr = queue->front;
queue->front = queue->front->next;
free (deletePtr);
} // while ในขณะที่
free (queue);
} // if ถ้า
return NULL;
} // destroyQueue ท้ายคิว

/* ===== printQueue =====พิมพ์คิว=====
A non-standard function that prints a queue. It is
non-standard because it accesses the queue structures.
Pre queue is a valid queue
Post queue data printed, front to rear
*/
void printQueue(QUEUE* queue)
{
// Local Definitions คำนิยามเฉพาะที่
QUEUE_NODE* node = queue->front;
// Statements คำสั่ง
printf ("Front=>");
while (node)
{
printf ("%3d", *(int*)node->dataPtr);
node = node->next;
} // while

```

```
printf(" <=Rear\n");  
return;  
} // printQueue พิมพ์คิว
```