```
#include <stdio.h>
#include <stdlib.h> // required for malloc()
//  Queue ADT Type Defintions ประเภทของคิว
   typedef struct node
    {
     void*       dataPtr;
     struct node* next;
    } QUEUE_NODE;
   typedef struct
    {
     QUEUE_NODE* front;
     QUEUE_NODE* rear;
     int      count;
    } QUEUE;

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

   bool  dequeue   (QUEUE* queue, void** itemPtr); // * = pointer พอยน์เตอร์
   bool  enqueue   (QUEUE* queue, void*  itemPtr); // ** = pointer of pointer พอยน์เตอร์ชี้พอยน์เตอร์
   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 จุดสิ้นสุดของคิว

void printQueue   (QUEUE* stack);

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

   // Create two queues สร้างคิวสองรายการ
   queue1 = createQueue();
   queue2 = createQueue();
   for (int i = 1; i <= 5; i++)
     {
      numPtr  = (int*)malloc(sizeof(i)); // set pointer to memory กำหนดพอยน์เตอร์ไปยังหน่วยความจำ
      *numPtr = i;
      enqueue(queue1, numPtr);
      if (!enqueue(queue2, numPtr))
        {
         printf ("\n\a**Queue overflow\n\n");
         exit (100);
        } // if !enqueue
```

```
    } // for สำหรับ

   printf ("Queue 1:\n");
   printQueue (queue1); // 1 2 3 4 5
   printf ("Queue 2:\n");
   printQueue (queue2); // 1 2 3 4 5
   return 0;
}


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


/*  แทรกสมาชิกต่อท้ายคิว */
bool enqueue (QUEUE* queue, void* itemPtr)
```

```
{
// Local Definitions นิยามท้องถิ่น
// QUEUE_NODE* newPtr; ห้าคิวโหนดชี้ไปยัง newPtr
// Statements
// 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

/* ดึงข้อมูลจากหัวคิวออกมาใช้งาน พร้อมห้าข้อมูลออกไปจากคิว */
bool dequeue (QUEUE* queue, void** itemPtr)
{
// Local Definitions
   QUEUE_NODE* deleteLoc;
// Statements
   if (!queue->count)
      return false;
```

```
   *itemPtr  = queue->front->dataPtr;
   deleteLoc = queue->front;
   if (queue->count == 1)
     // Deleting only item in queue ก็ลบเฉพาะรายการในคิว
     queue->rear  = queue->front = NULL;
   else
     queue->front = queue->front->next;
   (queue->count)--;
   free (deleteLoc);
   return true;
}  // dequeue

/* ดึงข้อมูลหัวคิวมาใช้งาน */
bool queueFront (QUEUE* queue, void** itemPtr)
{
//  Statements
   if (!queue->count)
      return false;
   else
     {
      *itemPtr = queue->front->dataPtr;
       return true;
     } // else
}  // queueFront

/* ดึงข้อมูลท้ายคิวมาใช้งาน*/
bool queueRear (QUEUE* queue, void** itemPtr)
```

```
{
//  Statements
   if (!queue->count)
      return true;
   else
     {
      *itemPtr = queue->rear->dataPtr;
      return false;
     } // else
}  // queueRear
```

/* ตรวจสอบค่าคิวว่างหรือไม่ */

```
bool emptyQueue (QUEUE* queue)
{
//  Statements
   return (queue->count == 0);
}  // emptyQueue
```

/* ตรวจสอบค่าค่าคิวเต็มหรือไม่ */

```
bool fullQueue (QUEUE* queue)
{
//  Check empty
if(emptyQueue(queue)) return false; // Not check in heap
//  Local Definitions *
QUEUE_NODE* temp;
//  Statements ชุดคำสั่ง
   temp = (QUEUE_NODE*)malloc(sizeof(*(queue->rear)));
```

```
   if (temp)
     {
      free (temp);
      return false; // Heap not full ไม่เต็ม
     } // if  ก้า
   return true; // Heap full  ปริมาณเต็ม
}  // fullQueue

/* วิเทิร์นค่าที่อยู่ในคิว */
int queueCount(QUEUE* queue)
{
//  Statements
   return queue->count;
}  // queueCount

/*  ทำลายคิว */
QUEUE* destroyQueue (QUEUE* queue)
{
//  Local Definitions เฉพาะที่
   QUEUE_NODE* deletePtr;
//  Statements ชุดคำสั่ง
   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 ทำลายคิว

/* พิมพ์คิว */
void printQueue(QUEUE* queue) {
//  Local Definitions เฉพาะที่
    QUEUE_NODE* node = queue->front;
//  Statements ชุดคำสั่ง
    printf ("Front=>");
    while (node)
      {
       printf ("%3d", *(int*)node->dataPtr);
       node = node->next;
      } // while ในขณะที่
    printf(" <=Rear\n");
    return;
}   // printQueue พิมพ์คิว
```