```
#include <stdio.h>
#include <stdlib.h> // required for malloc()
// Queue ADT Type Defintions คำจำกัดความประเภท ADT ของคิว
   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
   bool  enqueue   (QUEUE* queue, void*  itemPtr); // ** = pointer of pointer
   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 จุดสิ้นสุดของคำจำกัดความ ADT ของคิว

void printQueue   (QUEUE* stack);

int main (void)
{
```

```
// Local Definitions นิยามท้องถิ่น
   QUEUE* queue1;
   QUEUE* queue2;
   int*   numPtr;
   int** itemPtr;

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

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
   return queue;
}  // createQueue สร้างคิว

bool enqueue (QUEUE* queue, void* itemPtr)
{
// Local Definitions นิยามท้องถิ่น
// QUEUE_NODE* 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
     queue->front  = newPtr;
   else
     queue->rear->next = newPtr;
   (queue->count)++;
   queue->rear = newPtr;
   return true;
}  // enqueue

bool dequeue (QUEUE* queue, void** itemPtr)
{
```

```
// Local Definitions การกำหนด
   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
     queue->rear  = queue->front = NULL;
   else
     queue->front = queue->front->next;
   (queue->count)--;
   free (deleteLoc);
   return true;
}   // dequeue

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

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

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

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

bool fullQueue (QUEUE* queue)
{
//  Check empty
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
       } // if
    return true; // Heap full
}   // fullQueue

int queueCount(QUEUE* queue)
```

```
{
//  Statements
   return queue->count;
}   // queueCount

QUEUE* destroyQueue (QUEUE* queue)
{
//  Local Definitions
   QUEUE_NODE* deletePtr;
//  Statements
   if (queue)
      {
       while (queue->front != NULL)
          {
           free (queue->front->dataPtr);
           deletePtr   = queue->front;
           queue->front = queue->front->next;
           free (deletePtr);
          } // while
       free (queue);
      } // if
   return NULL;
}   // destroyQueue

void printQueue(QUEUE* queue)
{
//  Local Definitions
   QUEUE_NODE* node = queue->front;
//  Statements
   printf ("Front=>");
   while (node)
      {
```

```
        printf ("%3d", *(int*)node->dataPtr);
        node = node->next;
       } // while
    printf(" <=Rear\n");
    return;
}   // printQueue
```