```
/*
Using the queue ADT
edit from http://www.dreamincode.net/forums/topic/49439-concatenating-queues-in-c/
bin>bcc32 queue.cpp
*/
#include <stdio.h>
#include <stdlib.h> // required for malloc()
// Queue ADT Type Defintions กำหนดชนิดตัวแปลคิว
   typedef struct node
     {
      void*       dataPtr;
      struct node* next;
     } QUEUE_NODE;
   typedef struct
     {
      QUEUE_NODE* front;
      QUEUE_NODE* rear;
      int       count;
     } QUEUE;

// Prototype Declarations การประกาศตัวแปร
   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 จบการกำหนดชนิดคิว
```

```
void printQueue   (QUEUE* stack);

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

//  Statements คำสั่ง
   // Create two queues สร้างคิว 2 ตัว
   queue1 = createQueue();
   queue2 = createQueue();
   for (int i = 1; i <= 5; i++)
     {
      numPtr  = (int*)malloc(sizeof(i)); // set pointer to memory ตั้งตัวชี้ไปยังหน่วยความจำ
      *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;
}
```

```
QUEUE* createQueue (void)
{
// Local Definitions นิยามทั่วไป
   QUEUE* queue;
// Statements คำสั่ง
   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 นิยามทั่วไป
// QUEUE_NODE* 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  กำลังแทรกลงในคิวว่าง
     queue->front  = newPtr;
   else
     queue->rear->next = newPtr;
   (queue->count)++;
   queue->rear = newPtr;
```

```
   return true;
}   // enqueue

bool dequeue (QUEUE* queue, void** itemPtr)
{
//  Local Definitions นิยามทั่วไป
   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 กำลังลบเฉพาะรายการในคิว
     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  คำสั่ง
   if (!queue->count)
      return false;
   else
     {
      *itemPtr = queue->front->dataPtr;
       return true;
     } // else
```

```
}   // queueFront  ด้านหน้าคิว

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

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

bool fullQueue (QUEUE* queue)
{
//  Check empty ทำเครื่องหมายที่ว่างเปล่า
if(emptyQueue(queue)) return false; // Not check in heap
//  Local Definitions * นิยามทั่วไป
QUEUE_NODE* temp;
//  Statements  คำสั่ง
   temp = (QUEUE_NODE*)malloc(sizeof(*(queue->rear)));
   if (temp)
      {
       free (temp);
       return false; // Heap not full กองไม่ว่าง
```

```
      } // if
   return true; // Heap full กองเต็ม
}   // fullQueue คิวเต็ม

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

QUEUE* destroyQueue (QUEUE* queue)
{
//  Local Definitions นิยามทั่วไป
   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
       free (queue);
      } // if
   return NULL;
}   // destroyQueue ลบคิว

void printQueue(QUEUE* queue)
{
//  Local Definitions นิยามทั่วไป
```

```
    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
```